

LinuxCNC V2.9.0~pre0, 27 Aug 2022

Contents

I	Contents	1
II	About LinuxCNC	2
1	Introduction	3
2	LinuxCNC History	4
3	Origin	5
3.1	Name Change	6
3.2	Additional Info	6
III	Using LinuxCNC	7
4	General Info	8
4.1	User Foreword	8
4.2	LinuxCNC User Introduction	9
4.2.1	Introduction	9
4.2.2	How LinuxCNC Works	9
4.2.3	Graphical User Interfaces	11
4.2.4	Virtual Control Panels	18
4.2.5	Languages	19
4.2.6	Think Like a CNC Operator	19
4.2.7	Modes of Operation	19
4.3	Important User Concepts	20
4.3.1	Trajectory Control	20
4.3.1.1	Trajectory Planning	20
4.3.1.2	Path Following	20
4.3.1.3	Programming the Planner	21
4.3.1.4	Planning Moves	22

4.3.2	G-code	22
4.3.2.1	Defaults	22
4.3.2.2	Feed Rate	23
4.3.2.3	Tool Radius Offset	23
4.3.3	Homing	23
4.3.4	Tool Changes	23
4.3.5	Coordinate Systems	23
4.3.5.1	G53 Machine Coordinate	23
4.3.5.2	G54-59.3 User Coordinates	24
4.3.5.3	When You Are Lost	24
4.3.6	Machine Configurations	24
4.4	Starting LinuxCNC	26
4.4.1	Running LinuxCNC	26
4.4.1.1	Configuration Selector	27
4.5	CNC Machine Overview	28
4.5.1	Mechanical Components	28
4.5.1.1	Axes	28
4.5.1.2	Spindle	28
4.5.1.3	Coolant	28
4.5.1.4	Feed and Speed Override	29
4.5.1.5	Block Delete Switch	29
4.5.1.6	Optional Program Stop Switch	29
4.5.2	Control and Data Components	29
4.5.2.1	Linear Axes	29
4.5.2.2	Rotational Axes	29
4.5.2.3	Controlled Point	29
4.5.2.4	Coordinated Linear Motion	30
4.5.2.5	Feed Rate	30
4.5.2.6	Cooling	30
4.5.2.7	Dwell	30
4.5.2.8	Units	30
4.5.2.9	Current Position	31
4.5.2.10	Selected Plane	31
4.5.2.11	Tool Carousel	31
4.5.2.12	Tool Change	31
4.5.2.13	Pallet Shuttle	31
4.5.2.14	Speed Override	31
4.5.2.15	Path Control Mode	31
4.5.3	Interpreter Interaction with Switches	32

4.5.3.1	Feed and Speed Override Switches	32
4.5.3.2	Block Delete Switch	32
4.5.3.3	Optional Program Stop Switch	32
4.5.4	Tool Table	32
4.5.5	Parameters	32
4.6	Running LinuxCNC	33
4.6.1	Invoking LinuxCNC	33
4.6.2	Configuration Launcher	33
4.6.3	Next steps in configuration	36
4.6.4	Simulator Configurations	36
4.6.5	Configuration Resources	37
4.7	Stepper Configuration Wizard	37
4.7.1	Introduction	37
4.7.2	Start Page	38
4.7.3	Basic Information	39
4.7.4	Latency Test	40
4.7.5	Parallel Port Setup	42
4.7.6	Parallel Port 2 Setup	43
4.7.7	Axis Configuration	44
4.7.7.1	Finding Maximum Velocity	46
4.7.7.2	Finding Maximum Acceleration	47
4.7.8	Spindle Configuration	47
4.7.8.1	Spindle Speed Control	48
4.7.8.2	Spindle-synchronized motion	48
4.7.8.3	Determining Spindle Calibration	48
4.7.9	Options	50
4.7.10	Complete Machine Configuration	51
4.7.11	Axis Travels and Homes	51
4.7.11.1	Operating without Limit Switches	52
4.7.11.2	Operating without Home Switches	52
4.7.11.3	Home and Limit Switch wiring options	52
4.8	Mesa Configuration Wizard	53
4.8.1	Step by Step Instructions	54
4.8.2	Create or Edit	54
4.8.3	Basic Machine Information	55
4.8.4	External Configuration	58
4.8.5	GUI Configuration	59
4.8.6	Mesa Configuration	62
4.8.7	Mesa I/O Setup	64

4.8.8 Parallel port configuration	68
4.8.9 Axis Configuration	69
4.8.10 Spindle Configuration	76
4.8.11 Advanced Options	78
4.8.12 HAL Components	79
4.8.13 Advanced Usage Of PnCconf	80
4.9 Linux FAQ	82
4.9.1 Automatic Login	82
4.9.1.1 Debian	82
4.9.1.2 Ubuntu	82
4.9.2 Automatic Startup	83
4.9.3 Terminal	83
4.9.4 Man Pages	83
4.9.5 List Modules	83
4.9.6 Editing a Root File	84
4.9.6.1 The Command Line Way	84
4.9.6.2 The GUI Way	84
4.9.6.3 Root Access	84
4.9.7 Terminal Commands	84
4.9.7.1 Working Directory	84
4.9.7.2 Changing Directories	84
4.9.7.3 Listing files in a directory	85
4.9.7.4 Finding a File	85
4.9.7.5 Searching for Text	85
4.9.7.6 Diagnostic Messages	85
4.9.8 Convenience Items	86
4.9.8.1 Terminal Launcher	86
4.9.9 Hardware Problems	86
4.9.9.1 Hardware Info	86
4.9.9.2 Monitor Resolution	86
4.9.10 Paths	86
4.10 Lathe User Information	86
4.10.1 Lathe Mode	87
4.10.2 Lathe Tool Table	87
4.10.3 Lathe Tool Orientation	87
4.10.4 Tool Touch Off	89
4.10.4.1 X Touch Off	89
4.10.4.2 Z Touch Off	89
4.10.4.3 The Z Machine Offset	90

4.10.5	Spindle Synchronized Motion	90
4.10.6	Arcs	90
4.10.6.1	Arcs and Lathe Design	91
4.10.6.2	Radius & Diameter Mode	91
4.10.7	Tool Path	91
4.10.7.1	Control point	91
4.10.7.2	Cutting Angles without Cutter Comp	92
4.10.7.3	Cutting a Radius	93
4.10.7.4	Using Cutter Comp	95
4.11	Plasma Cutting Primer for LinuxCNC Users	95
4.11.1	What Is Plasma?	95
4.11.2	Arc Initialisation	96
4.11.2.1	High Frequency Start	96
4.11.2.2	Blowback Start	97
4.11.3	CNC Plasma	97
4.11.4	Choosing a Plasma Machine for CNC operations	98
4.11.5	Types Of Torch Height Control	99
4.11.6	Arc OK Signal	99
4.11.7	Initial Height Sensing	100
4.11.7.1	Float Switches	100
4.11.7.2	Ohmic Sensing	100
4.11.7.3	Hypersensing with a MESA THCAD-5	101
4.11.7.4	Example HAL Code for Hypersensing	102
4.11.8	THC Delay	103
4.11.9	Torch Voltage Sampling	103
4.11.10	Torch Breakaway	103
4.11.11	Corner Lock / Velocity Anti-Dive	104
4.11.12	Void / Kerf Crossing	104
4.11.13	Hole And Small Shape Cutting	104
4.11.14	I/O Pins For Plasma Controllers	105
4.11.14.1	Arc OK (input)	105
4.11.14.2	Torch On (output)	106
4.11.14.3	Float switch (input)	106
4.11.14.4	Ohmic Sensor enable (output)	106
4.11.14.5	Ohmic Sensing (input)	106
4.11.14.6	Torch Breakaway Sensor	107
4.11.15	G-Code For Plasma Controllers	107
4.11.15.1	Select Material Settings In QtPlasmaC And Use The Feedrate For That Material:	107

4.11.15.Enable/Disable THC Operation:	107
4.11.15.Reduce Cutting Speeds: (eg for hole cutting)	107
4.11.15.Cutter Compensation:	108
4.11.16.External Offsets and Plasma Cutting	108
4.11.17.Reading Arc Voltage With The Mesa THCAD	108
4.11.17.17.THCAD Connections	109
4.11.17.2.THCAD Initial Testing	109
4.11.17.3.Which Model THCAD To Use	110
4.11.18.Best Processors And Nesting	110
4.11.19.Designing For Noisy Electrical Environments	111
4.11.20.Water Tables	111
4.11.21.Downdraft Tables	112
4.11.22.Designing For Speed And Acceleration	112
4.11.23.Distance Travelled Per Motor Revolution	112
4.11.24.QtPlasmaC LinuxCNC Plasma Configuration	112
4.11.25.Hypertherm RS485 Control	112
4.11.26.Best Processors For Plasma Cutting	113
5 User Interfaces	114
5.1 AXIS GUI	114
5.1.1 Introduction	114
5.1.2 Getting Started	115
5.1.2.1 INI settings	115
5.1.2.2 A Typical Session	116
5.1.3 AXIS Window	116
5.1.3.1 Menu Items	117
5.1.3.2 Toolbar buttons	120
5.1.3.3 Graphical Display Area	121
5.1.3.4 Text Display Area	123
5.1.3.5 Manual Control	123
5.1.3.6 MDI	125
5.1.3.7 Feed Override	126
5.1.3.8 Spindle Speed Override	126
5.1.3.9 Jog Speed	126
5.1.3.10Max Velocity	127
5.1.4 Keyboard Controls	127
5.1.4.1 Feed Override Keys	127
5.1.5 Show LinuxCNC Status (linuxcnc_top)	128
5.1.6 MDI interface	129

5.1.7	axis-remote	130
5.1.8	Manual Tool Change	130
5.1.9	Python modules	130
5.1.10	Using AXIS in Lathe Mode	131
5.1.11	Using AXIS in Foam Cutting mode	133
5.1.12	Advanced Configuration	134
5.1.12.1	Program Filters	135
5.1.12.2	The X Resource Database	136
5.1.12.3	Jogwheel	136
5.1.12.4	~/.axisrc	136
5.1.12.5	USER_COMMAND_FILE	137
5.1.12.6	user_live_update()	137
5.1.12.7	user_hal_pins()	137
5.1.12.8	External Editor	137
5.1.12.9	Virtual Control Panel	137
5.1.12.10	Preview Control	137
5.1.13	Axisui	138
5.1.14	AXIS Customization Hints	138
5.1.14.1	The update function	139
5.1.14.2	Disable the Close Dialog	139
5.1.14.3	Change the Text Font	139
5.1.14.4	Modify Rapid Rate with Keyboard Shortcuts	139
5.1.14.5	Read the INI file	140
5.1.14.6	Read linuxcnc Status	140
5.1.14.7	Change the current view	140
5.1.14.8	Creating new AXISUI HAL Pins	140
5.1.14.9	Creating new HAL Component and Pins	140
5.1.14.10	Switch Tabs with HAL Pins	141
5.1.14.11	Add a GOTO Home button	141
5.1.14.12	Add Button to manual frame	142
5.1.14.13	Reading Internal Variables	142
5.1.14.14	Hide Widgets	143
5.1.14.15	Change a label	143
5.1.14.16	Redirect an existing command	144
5.1.14.17	Change the DRO color	144
5.1.14.18	Change the Toolbar Buttons	144
5.1.14.19	Change Plotter Colors	145
5.2	GMOCCAPY	145
5.2.1	Introduction	145

5.2.2	Requirements	146
5.2.3	How To Get GMOCCAPY	147
5.2.4	Basic Configuration	147
5.2.4.1	The DISPLAY Section	148
5.2.4.2	The RS274NGC Section	156
5.2.4.3	The MACRO Section	156
5.2.4.4	The TRAJ Section	159
5.2.5	HAL Pins	159
5.2.5.1	Right And Bottom Button Lists	160
5.2.5.2	Velocities And Overrides	164
5.2.5.3	Jog HAL Pins	167
5.2.5.4	Jog Velocities And Turtle-Jog HAL Pin	167
5.2.5.5	Jog Increment HAL Pins	168
5.2.5.6	Hardware Unlock Pin	168
5.2.5.7	Error/Warning Pins	168
5.2.5.8	User Created Message HAL Pins	169
5.2.5.9	Spindle Feedback Pins	169
5.2.5.10	Pins To Indicate Program Progress Information	170
5.2.5.11	Tool related pins	170
5.2.6	Auto Tool Measurement	171
5.2.6.1	Tool Measurement Pins	172
5.2.6.2	Tool Measurement INI File Modifications	173
5.2.6.3	Needed Files	174
5.2.6.4	Needed HAL Connections	174
5.2.7	The Settings Page	174
5.2.7.1	Appearance	175
5.2.7.2	Hardware	180
5.2.7.3	Advanced Settings	182
5.2.8	Icon Theme	184
5.2.8.1	Custom Icon Theme	184
5.2.8.2	Symbolic Icons	185
5.2.9	Lathe Specific Section	186
5.2.10	Plasma Specific Section	189
5.2.11	Videos On YouTube	189
5.2.11.1	Basic Usage	190
5.2.11.2	Simulated Jog Wheels	190
5.2.11.3	Settings Page	190
5.2.11.4	Simulated Hardware Button	190
5.2.11.5	User Tabs	190

5.2.11.6Tool Measurement Videos	190
5.2.12Known problems	190
5.2.12.1Strange numbers in the info area	190
5.2.12.2Not ending macro	191
5.3 NGCGUI	192
5.3.1 Overview	192
5.3.2 Demonstration Configurations	193
5.3.3 Library Locations	195
5.3.4 Standalone Usage	196
5.3.4.1 Standalone NGCGUI	196
5.3.4.2 Standalone PYNGCGUI	196
5.3.5 Embedding NGCGUI	197
5.3.5.1 Embedding NGCGUI in AXIS	197
5.3.5.2 Embedding PYNGCGUI as a GladeVCP tab page in a GUI	198
5.3.5.3 Additional INI File items required for ngcgui or pyngcgui	198
5.3.5.4 Truetype Tracer	200
5.3.5.5 INI File Path Specifications	200
5.3.5.6 Summary of INI File item details for NGCGUI usage	201
5.3.6 File Requirements for NGCGUI Compatibility	203
5.3.6.1 Single-File Gcode (.ngc) Subroutine Requirements	203
5.3.6.2 Gcode-meta-compiler (.gcmc) file requirements	205
5.3.7 DB25 Example	206
5.3.8 Creating a subroutine	209
5.4 The Touchy Graphical User Interface	210
5.4.1 Panel Configuration	210
5.4.1.1 HAL connections	210
5.4.1.2 Recommended for any setup	211
5.4.2 Setup	211
5.4.2.1 Enabling Touchy	211
5.4.2.2 Preferences	212
5.4.2.3 Macros	212
5.5 Gscreen	212
5.5.1 Introduction	212
5.5.1.1 Glade File	217
5.5.1.2 PyGTK	217
5.5.2 GladeVCP	218
5.5.2.1 Overview	218
5.5.2.2 Build a GladeVCP Panel	219
5.5.3 Building a simple clean-sheet custom screen	220

5.5.4	Handler file example	222
5.5.4.1	Adding Keybindings Functions	223
5.5.4.2	Linuxcnc State Status	223
5.5.4.3	Jogging Keys	224
5.5.5	Gscreen Start Up	224
5.5.6	INI Settings	226
5.5.7	User Dialog Messages	226
5.5.7.1	Copy the Stock Handler/Glade File For Modification	227
5.6	TkLinuxCNC GUI	227
5.6.1	Introduction	227
5.6.2	Getting Started	228
5.6.2.1	A typical session with TkLinuxCNC	228
5.6.3	Elements of the TkLinuxCNC window	229
5.6.3.1	Main buttons	229
5.6.3.2	Offset display status bar	229
5.6.3.3	Coordinate Display Area	230
5.6.3.4	TkLinuxCNC Interpreter / Automatic Program Control	230
5.6.3.5	Manual Control	230
5.6.3.6	Code Entry	231
5.6.3.7	Jog Speed	232
5.6.3.8	Feed Override	232
5.6.3.9	Spindle speed Override	232
5.6.4	Keyboard Controls	232
5.7	QtPlasmaC	232
5.7.1	Preamble	232
5.7.2	License	233
5.7.3	Introduction	233
5.7.4	Installing LinuxCNC	236
5.7.4.1	If The User Does Not Have Linux Installed	236
5.7.4.2	Package Installation (Buildbot) If The User Has Linux with LinuxCNC v2.8	237
5.7.4.3	Run In Place Installation If The User Has Linux with LinuxCNC v2.8	237
5.7.5	Creating A QtPlasmaC Configuration	237
5.7.5.1	Modes	237
5.7.5.2	Available I/Os	238
5.7.5.3	Recommended Settings:	239
5.7.5.4	Configuring	240
5.7.5.5	Qt Dependency Errors	245
5.7.5.6	Initial Setup	245
5.7.6	Migrating to QtPlasmac From PlasmaC (Axis or Gmoccapy)	248

5.7.6.1 Quick Method	248
5.7.6.2 New Base Config Method	250
5.7.7 Other QtPlasmaC Setup Considerations	251
5.7.7.1 Lowpass Filter	251
5.7.7.2 Contact Bounce	251
5.7.7.3 Contact Load	252
5.7.7.4 Desktop Launcher	253
5.7.7.5 QtPlasmaC Files	254
5.7.7.6 INI File	254
5.7.8 QtPlasmaC GUI Overview	256
5.7.8.1 Exiting QtPlasmaC	256
5.7.8.2 MAIN Tab	257
5.7.8.3 Preview Views	262
5.7.8.4 CONVERSATIONAL Tab	263
5.7.8.5 PARAMETERS Tab	263
5.7.8.6 STATISTICS Tab	269
5.7.9 Using QtPlasmaC	270
5.7.9.1 Units Systems	271
5.7.9.2 Preamble and Postamble Codes	271
5.7.9.3 Mandatory Codes	271
5.7.9.4 Coordinates	272
5.7.9.5 Cut Feed Rate	272
5.7.9.6 Material File	272
5.7.9.7 Manual Material Handling	274
5.7.9.8 Automatic Material Handling	274
5.7.9.9 Material Addition Via Magic Comments In G-code	275
5.7.9.10Material Converter	276
5.7.9.11CAMERA	279
5.7.9.12LASER	280
5.7.9.13Path Tolerance	281
5.7.9.14Paused Motion	282
5.7.9.15Pause At End Of Cut	282
5.7.9.16Multiple Tools	282
5.7.9.17Velocity Reduction	283
5.7.9.18THC (Torch Height Controller)	284
5.7.9.19Cutter Compensation	285
5.7.9.20Initial Height Sense (IHS) Skip	285
5.7.9.21Probing	285
5.7.9.22Offset Probing	286

5.7.9.23Cut Types	287
5.7.9.24Hole Cutting - Intro	287
5.7.9.25Hole Cutting	288
5.7.9.26Hole Cutting - Automatic	289
5.7.9.27Single Cut	290
5.7.9.28Thick Materials	292
5.7.9.29Mesh Mode (Expanded Metal Cutting)	293
5.7.9.30Ignore Arc OK	293
5.7.9.31Cut Recovery	294
5.7.9.32Run From Line	295
5.7.9.33Scribe	296
5.7.9.34Spotting	298
5.7.9.35Virtual Keyboard Custom Layouts	299
5.7.9.36Keyboard Shortcuts	300
5.7.9.37MDI	302
5.7.10Conversational Shape Library	302
5.7.10.1Conversational Settings	304
5.7.10.2Conversational Lines And Arcs	305
5.7.10.3Conversational Single Shape	306
5.7.10.4Conversational Group Of Shapes	307
5.7.10.5Conversational Block	307
5.7.10.6Conversational Saving A Job	308
5.7.11Error Messages	308
5.7.11.1Error Logging	308
5.7.11.2Error Message Display	309
5.7.11.3Critical Errors	309
5.7.11.4Warning Errors	310
5.7.12Updating QtPlasmaC	311
5.7.12.1Standard Update	311
5.7.12.2Continuous Update	312
5.7.13Modify An Existing QtPlasmaC Configuration	312
5.7.14Customizing QtPlasmaC GUI	312
5.7.14.1Add A Custom Style	312
5.7.14.2Create A New Style	313
5.7.14.3Returning To The Default Styling	313
5.7.15QtPlasmaC Advanced Topics	314
5.7.15.1Custom User Buttons	314
5.7.15.2Peripheral Offsets (Laser, Camera, Scribe, Offset Probe)	320
5.7.15.3Keep Z Motion	321

5.7.15.4	External HAL Pins	321
5.7.15.5	Hide Program Buttons	322
5.7.15.6	Tuning Void Sensing	323
5.7.15.7	Lost Arc Delay	323
5.7.15.8	Tuning Mode 0 Arc OK	323
5.7.15.9	Zero Window	324
5.7.15.10	Max Offset	324
5.7.15.11	Enable Tabs During Automated Motion	324
5.7.15.12	Override Jog Inhibit Via Z+ Jog	324
5.7.15.13	QtPlasmaC State Outputs	325
5.7.15.14	QtPlasmaC Debug Print	325
5.7.15.15	Hypertherm PowerMax Communications	325
5.7.16	Internationalisation	327
5.7.17	Appendix	328
5.7.17.1	Example Configurations	328
5.7.17.2	NGC Samples	328
5.7.17.3	QtPlasmaC Specific G-codes	329
5.7.17.4	QtPlasmaC G-code Examples	330
5.7.17.5	Mesa THCAD	332
5.7.17.6	RS485 Connections	334
5.7.17.7	Arc OK With A Reed Relay	336
5.7.17.8	Contact Load Schematics	338
5.7.18	Known Issues	338
5.7.18.1	Keyboard Jogging	338
5.7.19	Support	339
5.8	MDRO GUI	339
5.8.1	Introduction	339
5.8.2	Getting Started	340
5.8.2.1	INI File Options	340
5.8.2.2	Command Line Options	341
5.8.2.3	Pins	341
5.8.3	MDRO Window	341
5.8.4	Index operations	342
5.8.5	Simulation	342

6 Programming	343
6.1 Coordinate Systems	343
6.1.1 Introduction	343
6.1.2 Machine Coordinate System	343
6.1.2.1 Machine coordinates moves: G53	343
6.1.3 Coordinate Systems	344
6.1.3.1 Default Coordinate System	345
6.1.3.2 Setting Coordinate System Offsets	346
6.1.4 Local and Global Offsets	346
6.1.4.1 The G52 command	346
6.1.5 G92 Axes Offsets	347
6.1.5.1 The G92 commands	347
6.1.5.2 Setting G92 Values	348
6.1.5.3 G92 Persistence Cautions	348
6.1.5.4 G92 and G52 Interaction Cautions	349
6.1.6 Sample Programs Using Offsets	349
6.1.6.1 Sample Program Using Workpiece Coordinate Offsets	349
6.1.6.2 Sample Program Using G52 Offsets	350
6.2 G-Code Overview	350
6.2.1 Overview	350
6.2.2 Format of a line	351
6.2.2.1 /: Block Delete	351
6.2.2.2 Line Number	351
6.2.2.3 Word	352
6.2.2.4 Numbers	352
6.2.3 Parameters	353
6.2.3.1 Numbered Parameters	354
6.2.3.2 Subroutine Parameters	356
6.2.3.3 Named Parameters	356
6.2.3.4 Predefined Named Parameters	357
6.2.3.5 System Parameters	358
6.2.4 HAL pins and INI values	360
6.2.5 Expressions	361
6.2.6 Binary Operators	361
6.2.7 Equality and floating-point values	361
6.2.8 Functions	361
6.2.9 Repeated Items	362
6.2.10 Item order	363
6.2.11 Commands and Machine Modes	363

6.2.12Polar Coordinates	363
6.2.13Modal Groups	365
6.2.14Comments	366
6.2.15Messages	367
6.2.16Probe Logging	367
6.2.17Logging	367
6.2.18Debug Messages	368
6.2.19Print Messages	368
6.2.20Comment Parameters	368
6.2.21File Requirements	368
6.2.22File Size	369
6.2.23G-Code Order of Execution	369
6.2.24G-Code Best Practices	370
6.2.25Linear and Rotary Axis	371
6.2.26Common Error Messages	371
6.3 G-Codes	371
6.3.1 Conventions	371
6.3.2 G-Code Quick Reference Table	372
6.3.3 G0 Rapid Move	373
6.3.3.1 Rapid Velocity Rate	373
6.3.4 G1 Linear Move	373
6.3.5 G2, G3 Arc Move	374
6.3.5.1 Center Format Arcs	375
6.3.5.2 Center Format Examples	376
6.3.5.3 Radius Format Arcs	378
6.3.6 G4 Dwell	379
6.3.7 G5 Cubic Spline	379
6.3.8 G5.1 Quadratic Spline	380
6.3.9 G5.2 G5.3 NURBS Block	381
6.3.10G7 Lathe Diameter Mode	382
6.3.11G8 Lathe Radius Mode	382
6.3.12G10 L0 Reload Tool Table Data	383
6.3.13G10 L1 Set Tool Table	383
6.3.14G10 L2 Set Coordinate System	384
6.3.15G10 L10 Set Tool Table	385
6.3.16G10 L11 Set Tool Table	386
6.3.17G10 L20 Set Coordinate System	386
6.3.18G17 - G19.1 Plane Select	387
6.3.19G20, G21 Units	387

6.3.20G28, G28.1 Go/Set Predefined Position	387
6.3.21G30, G30.1 Go/Set Predefined Position	388
6.3.22G33 Spindle Synchronized Motion	388
6.3.23G33.1 Rigid Tapping	389
6.3.24G38.n Straight Probe	390
6.3.25G40 Compensation Off	392
6.3.26G41, G42 Cutter Compensation	392
6.3.27G41.1, G42.1 Dynamic Cutter Compensation	393
6.3.28G43 Tool Length Offset	393
6.3.29G43.1 Dynamic Tool Length Offset	394
6.3.30G43.2 Apply additional Tool Length Offset	394
6.3.31G49 Cancel Tool Length Compensation	395
6.3.32G52 Local Coordinate System Offset	395
6.3.33G53 Move in Machine Coordinates	395
6.3.34G54-G59.3 Select Coordinate System	395
6.3.35G61 Exact Path Mode	396
6.3.36G61.1 Exact Stop Mode	396
6.3.37G64 Path Blending	396
6.3.38G70 Lathe finishing cycle	397
6.3.39G71 G72 Lathe roughing cycles	398
6.3.40G73 Drilling Cycle with Chip Breaking	399
6.3.41G74 Left-hand Tapping Cycle with Dwell	399
6.3.42G76 Threading Cycle	400
6.3.43G80-G89 Canned Cycles	403
6.3.43.1Common Words	403
6.3.43.2Sticky Words	403
6.3.43.3Repeat Cycle	403
6.3.43.4Retract Mode	403
6.3.43.5Canned Cycle Errors	404
6.3.43.6Preliminary and In-Between Motion	404
6.3.43.7Why use a canned cycle?	404
6.3.44G80 Cancel Canned Cycle	406
6.3.45G81 Drilling Cycle	407
6.3.46G82 Drilling Cycle, Dwell	412
6.3.47G83 Peck Drilling Cycle	412
6.3.48G84 Right-hand Tapping Cycle, Dwell	413
6.3.49G85 Boring Cycle, Feed Out	413
6.3.50G86 Boring Cycle, Spindle Stop, Rapid Move Out	414
6.3.51G87 Back Boring Cycle	414

6.3.52G88 Boring Cycle, Spindle Stop, Manual Out	414
6.3.53G89 Boring Cycle, Dwell, Feed Out	414
6.3.54G90, G91 Distance Mode	414
6.3.55G90.1, G91.1 Arc Distance Mode	415
6.3.56G92 Coordinate System Offset	415
6.3.57G92.1, G92.2 Reset G92 Offsets	416
6.3.58G92.3 Restore G92 Offsets	416
6.3.59G93, G94, G95 Feed Rate Mode	416
6.3.60G96, G97 Spindle Control Mode	417
6.3.61G98, G99 Canned Cycle Return Level	417
6.4 M-Codes	418
6.4.1 M-Code Quick Reference Table	418
6.4.2 M0, M1 Program Pause	418
6.4.3 M2, M30 Program End	418
6.4.4 M60 Pallet Change Pause	419
6.4.5 M3, M4, M5 Spindle Control	419
6.4.6 M6 Tool Change	420
6.4.6.1 Manual Tool Change	420
6.4.6.2 Tool Changer	420
6.4.7 M7, M8, M9 Coolant Control	421
6.4.8 M19 Orient Spindle	421
6.4.9 M48, M49 Speed and Feed Override Control	422
6.4.10M50 Feed Override Control	422
6.4.11M51 Spindle Speed Override Control	422
6.4.12M52 Adaptive Feed Control	422
6.4.13M53 Feed Stop Control	422
6.4.14M61 Set Current Tool	423
6.4.15M62 - M65 Digital Output Control	423
6.4.16M66 Wait on Input	423
6.4.17M67 Analog Output,Synchronized	424
6.4.18M68 Analog Output, Immediate	425
6.4.19M70 Save Modal State	425
6.4.20M71 Invalidate Stored Modal State	426
6.4.21M72 Restore Modal State	426
6.4.22M73 Save and Autorestore Modal State	427
6.4.23M98 and M99	428
6.4.23.1Selectively Restoring Modal State	428
6.4.24M100-M199 User Defined Commands	428
6.5 O Codes	430

6.5.1	Use of O-codes	430
6.5.2	Numbering	430
6.5.3	Comments	430
6.5.4	Subroutines	431
6.5.4.1	Fanuc-Style Numbered Programs	432
6.5.5	Looping	433
6.5.6	Conditional	434
6.5.7	Repeat	435
6.5.8	Indirection	435
6.5.9	Calling Files	435
6.5.10	Subroutine return values	436
6.5.11	Errors	436
6.6	Other Codes	436
6.6.1	F: Set Feed Rate	436
6.6.2	S: Set Spindle Speed	436
6.6.3	T: Select Tool	437
6.7	G-Code Examples	437
6.7.1	Mill Examples	437
6.7.1.1	Helical Hole Milling	437
6.7.1.2	Slotting	438
6.7.1.3	Grid Probe	438
6.7.1.4	Smart Probe	439
6.7.1.5	Tool Length Probe	440
6.7.1.6	Hole Probe	440
6.7.1.7	Cutter Compensation	440
6.7.2	Lathe Examples	441
6.7.2.1	Threading	441
6.8	RS274/NGC Differences	441
6.8.1	Changes from RS274/NGC	441
6.8.2	Additions to RS274/NGC	442
6.9	Image to G-Code	443
6.9.1	What is a depth map?	443
6.9.2	Integrating image-to-gcode with the AXIS user interface	443
6.9.3	Using image-to-gcode	443
6.9.4	Option Reference	444
6.9.4.1	Units	444
6.9.4.2	Invert Image	444
6.9.4.3	Normalize Image	444
6.9.4.4	Expand Image Border	444

6.9.4.5 Tolerance (units)	444
6.9.4.6 Pixel Size (units)	444
6.9.4.7 Plunge Feed Rate (units per minute)	444
6.9.4.8 Feed Rate (units per minute)	444
6.9.4.9 Spindle Speed (RPM)	445
6.9.4.10 Scan Pattern	445
6.9.4.11 Scan Direction	445
6.9.4.12 Depth (units)	445
6.9.4.13 Step Over (pixels)	445
6.9.4.14 Tool Diameter	445
6.9.4.15 Safety Height	445
6.9.4.16 Tool Type	446
6.9.4.17 Lace bounding	446
6.9.4.18 Contact angle	446
6.9.4.19 Roughing offset and depth per pass	446
7 Tool Compensation	448
7.1 Tool Compensation	448
7.1.1 Touch Off	448
7.1.1.1 Using G10 L1/L10/L11	449
7.1.2 Tool Table	449
7.1.2.1 Tool Table Format	449
7.1.2.2 Tool IO	451
7.1.2.3 Tool Changers	452
7.1.3 Tool Length Compensation	453
7.1.4 Cutter Radius Compensation	453
7.1.4.1 Overview	454
7.1.4.2 Examples	456
7.2 Tool Edit GUI	457
7.2.1 Overview	457
7.2.2 Column Sorting	458
7.2.3 Columns Selection	459
7.2.4 Stand Alone Use	459
IV Configuration	461
8 General Info	462
8.1 Integrator Concepts	462
8.1.1 File Locations	462

8.1.1.1	Installed	462
8.1.1.2	Command Line	463
8.1.2	Files	463
8.1.3	Stepper Systems	463
8.1.3.1	Base Period	463
8.1.3.2	Step Timing	464
8.1.4	Servo Systems	464
8.1.4.1	Basic Operation	464
8.1.4.2	Proportional term	466
8.1.4.3	Integral term	466
8.1.4.4	Derivative term	466
8.1.4.5	Loop tuning	467
8.1.4.6	Manual tuning	467
8.1.5	RTAI	467
8.1.5.1	ACPI	467
8.2	Latency Testing	467
8.2.1	Latency Test	467
8.2.2	Latency Plot	469
8.2.3	Latency Histogram	470
8.3	Stepper Tuning	471
8.3.1	Getting the most out of Software Stepping	471
8.3.1.1	Run a Latency Test	471
8.3.1.2	Figure out what your drives expect	472
8.3.1.3	Choose your BASE_PERIOD	473
8.3.1.4	Use steplen, stepspace, dirsetup, and/or dirhold	473
8.3.1.5	No Guessing!	474
8.4	Stepper Diagnostics	474
8.4.1	Common Problems	474
8.4.1.1	Stepper Moves One Step	474
8.4.1.2	No Steppers Move	475
8.4.1.3	Distance Not Correct	475
8.4.2	Error Messages	475
8.4.2.1	Following Error	475
8.4.2.2	RTAPI Error	475
8.4.3	Testing	476
8.4.3.1	Step Timing	476

9 Configuration	477
9.1 Stepper Quickstart	477
9.1.1 Latency Test	477
9.1.2 Sherline	477
9.1.3 Xylotex	477
9.1.4 Machine Information	477
9.1.5 Pinout Information	478
9.1.6 Mechanical Information	478
9.2 INI Configuration	479
9.2.1 The INI File Components	479
9.2.1.1 Comments	480
9.2.1.2 Sections	480
9.2.1.3 Variables	481
9.2.1.4 Custom Sections and Variables	481
9.2.1.5 Include Files	482
9.2.2 INI File Sections	482
9.2.2.1 [EMC] Section	482
9.2.2.2 [DISPLAY] Section	483
9.2.2.3 [FILTER] Section	487
9.2.2.4 [RS274NGC] Section	488
9.2.2.5 [EMCMOT] Section	490
9.2.2.6 [TASK] Section	490
9.2.2.7 [HAL] section	490
9.2.2.8 [HALUI] section	492
9.2.2.9 [APPLICATIONS] Section	492
9.2.2.10[TRAJ] Section	492
9.2.2.11[KINS] Section	495
9.2.2.12[AXIS_<letter>] Section	495
9.2.2.13[JOINT_<num>] Sections	496
9.2.2.14[SPINDLE_<num>] Section(s)	502
9.2.2.15[EMCIO] Section	503
9.3 Homing Configuration	503
9.3.1 Overview	503
9.3.2 Prerequisite	504
9.3.3 Separate Home Switch Example Layout	505
9.3.4 Shared Limit/Home Switch Example Layout	506
9.3.5 Homing Sequence	506
9.3.6 Configuration	508
9.3.6.1 HOME_SEARCH_VEL	508

9.3.6.2	HOME_LATCH_VEL	508
9.3.6.3	HOME_FINAL_VEL	508
9.3.6.4	HOME_IGNORE_LIMITS	509
9.3.6.5	HOME_USE_INDEX	509
9.3.6.6	HOME_INDEX_NO_ENCODER_RESET	509
9.3.6.7	HOME_OFFSET	509
9.3.6.8	HOME	509
9.3.6.9	HOME_IS_SHARED	510
9.3.6.10	HOME_ABSOLUTE_ENCODER	510
9.3.6.11	HOME_SEQUENCE	510
9.3.6.12	VOLATILE_HOME	511
9.3.6.13	LOCKING_INDEXER	511
9.3.6.14	Immediate Homing	511
9.3.6.15	Inhibiting Homing	512
9.4	I/O Control V2	513
9.4.1	Description	513
9.4.2	Usage	513
9.4.3	Pins	514
9.4.4	Parameters	515
9.4.5	Communications	515
9.5	Lathe Configuration	516
9.5.1	Default Plane	516
9.5.2	INI Settings	516
9.6	Remap Extending G-code	517
9.6.1	Introduction: Extending the RS274NGC Interpreter by Remapping Codes	517
9.6.1.1	A Definition: Remapping Codes	517
9.6.1.2	Why would you want to extend the RS274NGC Interpreter?	517
9.6.2	Getting started	519
9.6.2.1	Builtin Remaps	519
9.6.2.2	Picking a code	520
9.6.2.3	Parameter handling	520
9.6.2.4	Handling results	521
9.6.2.5	Execution sequencing	521
9.6.2.6	An minimal example remapped code	521
9.6.3	Configuring Remapping	522
9.6.3.1	The REMAP statement	522
9.6.3.2	Useful REMAP option combinations	523
9.6.3.3	The argspec parameter	523
9.6.4	Upgrading an existing configuration for remapping	527

9.6.5	Remapping tool change-related codes: T, M6, M61	527
9.6.5.1	Overview	527
9.6.5.2	Understanding the role of iocontrol with remapped tool change codes . . .	528
9.6.5.3	Specifying the M6 replacement	529
9.6.5.4	Configuring iocontrol with a remapped M6	531
9.6.5.5	Writing the change and prepare O-word procedures	531
9.6.5.6	Making minimal changes to the built in codes, including M6	531
9.6.5.7	Specifying the T (prepare) replacement	532
9.6.5.8	Error handling: dealing with abort	533
9.6.5.9	Error handling: failing a remapped code NGC procedure	534
9.6.6	Remapping other existing codes: S, M0, M1, M60	535
9.6.6.1	Automatic gear selection be remapping S (set spindle speed)	535
9.6.6.2	Adjusting the behavior of M0, M1, M60	535
9.6.7	Creating new G-code cycles	535
9.6.8	Configuring Embedded Python	536
9.6.8.1	Python plugin : ini file configuration	536
9.6.8.2	Executing Python statements from the interpreter	537
9.6.9	Programming Embedded Python in the RS274NGC Interpreter	537
9.6.9.1	The Python plugin namespace	537
9.6.9.2	The Interpreter as seen from Python	537
9.6.9.3	The Interpreter <code>__init__</code> and <code>__delete__</code> functions	537
9.6.9.4	Calling conventions: NGC to Python	538
9.6.9.5	Calling conventions: Python to NGC	541
9.6.9.6	Built in modules	542
9.6.10	Adding Predefined Named Parameters	543
9.6.11	Standard Glue routines	543
9.6.11.1	T: prepare_prolog and prepare_epilog	544
9.6.11.2	M6: change_prolog and change_epilog	544
9.6.11.3	G-code Cycles: cycle_prolog and cycle_epilog	545
9.6.11.4	S (Set Speed) : setspeed_prolog and setspeed_epilog	546
9.6.11.5	F (Set Feed) : setfeed_prolog and setfeed_epilog	546
9.6.11.6	M61 Set tool number : settool_prolog and settool_epilog	546
9.6.12	Remapped code execution	546
9.6.12.1	NGC procedure call environment during remaps	546
9.6.12.2	Nested remapped codes	546
9.6.12.3	Sequence number during remaps	546
9.6.12.4	Debugging flags	547
9.6.12.5	Debugging Embedded Python code	547
9.6.13	Axis Preview and Remapped code execution	548

9.6.14	Remappable Codes	549
9.6.14.1	Existing codes which can be remapped	549
9.6.14.2	Currently unallocated G-codes:	549
9.6.14.3	Currently unallocated M-codes:	552
9.6.14.4	readahead time and execution time	553
9.6.14.5	plugin/pickle hack	553
9.6.14.6	Module, methods, classes, etc reference	553
9.6.15	Introduction: Extending Task Execution	553
9.6.15.1	Why would you want to change Task Execution?	553
9.6.15.2	A diagram: task, interp, iocontrol, UI (??)	553
9.6.16	Models of Task execution	553
9.6.16.1	Traditional iocontrol/iocontrolv2 execution	554
9.6.16.2	Redefining IO procedures	554
9.6.16.3	Execution-time Python procedures	554
9.6.17	A short survey of LinuxCNC program execution	554
9.6.17.1	Interpreter state	554
9.6.17.2	Task and Interpreter interaction, Queuing and Read-Ahead	554
9.6.17.3	Predicting the machine position	555
9.6.17.4	Queue-busters break position prediction	555
9.6.17.5	How queue-busters are dealt with	555
9.6.17.6	Word order and execution order	556
9.6.17.7	Parsing	556
9.6.17.8	Execution	556
9.6.17.9	Procedure execution	556
9.6.17.10	How tool change currently works	556
9.6.17.11	How Tx (Prepare Tool) works	557
9.6.17.12	How M6 (Change tool) works	557
9.6.17.13	How M61 (Change tool number) works	559
9.6.18	Status	559
9.6.19	Changes	559
9.6.20	Debugging	559
9.7	Moveoff Component	559
9.7.1	Modifying an existing configuration	561
9.8	Stepper Configuration	564
9.8.1	Introduction	564
9.8.2	Maximum step rate	564
9.8.3	Pinout	564
9.8.3.1	Standard Pinout HAL	565
9.8.3.2	Overview	566

9.8.3.3	Changing the standard_pinout.hal	566
9.8.3.4	Changing polarity of a signal	567
9.8.3.5	Adding PWM Spindle Speed Control	567
9.8.3.6	Adding an enable signal	567
9.8.3.7	External ESTOP button	567
10	Control Panels	568
10.1	PyVCP	568
10.1.1	Introduction	568
10.1.2	Panel Construction	569
10.1.3	Security	570
10.1.4	AXIS	570
10.1.5	Stand Alone	571
10.1.6	Widgets	572
10.1.6.1	Syntax	573
10.1.6.2	General Notes	573
10.1.6.3	Label	574
10.1.6.4	Multi_Label	574
10.1.6.5	LEDs	575
10.1.6.6	Buttons	576
10.1.6.7	Number Displays	578
10.1.6.8	Number Inputs	581
10.1.6.9	Images	585
10.1.6.10	Containers	587
10.2	PyVCP Examples	592
10.2.1	AXIS	592
10.2.2	Floating Panels	592
10.2.3	Jog Buttons Example	593
10.2.3.1	Create the Widgets	594
10.2.3.2	Make Connections	596
10.2.4	Port Tester	597
10.2.5	GS2 RPM Meter	600
10.2.5.1	The Panel	600
10.2.5.2	The Connections	602
10.2.6	Rapid to Home Button	602
10.3	GladeVCP: Glade Virtual Control Panel	604
10.3.1	What is GladeVCP?	604
10.3.1.1	PyVCP versus GladeVCP at a glance	604
10.3.2	A Quick Tour with the Example Panel	605

10.3.2.1	Exploring the example panel	608
10.3.2.2	Exploring the User Interface description	609
10.3.2.3	Exploring the Python callback	609
10.3.3	Creating and Integrating a Glade user interface	609
10.3.3.1	Prerequisite: Glade installation	609
10.3.3.2	Running Glade to create a new user interface	609
10.3.3.3	Testing a panel	610
10.3.3.4	Preparing the HAL command file	611
10.3.3.5	Integrating into AXIS, like PyVCP	611
10.3.3.6	Embedding as a Tab	612
10.3.3.7	Integrating into Touchy	612
10.3.4	GladeVCP command line options	613
10.3.5	Understanding the GladeVCP startup process	614
10.3.6	HAL Widget reference	615
10.3.6.1	Widget and HAL pin naming	615
10.3.6.2	Python attributes and methods of HAL Widgets	616
10.3.6.3	Setting pin and widget values	616
10.3.6.4	The hal-pin-changed signal	616
10.3.6.5	Buttons	617
10.3.6.6	Scales	618
10.3.6.7	SpinButton	618
10.3.6.8	Hal_Dial	619
10.3.6.9	Jog Wheel	621
10.3.6.10	Speed Control	622
10.3.6.11	Label	625
10.3.6.12	Containers	625
10.3.6.13	BED	626
10.3.6.14	ProgressBar	627
10.3.6.15	ComboBox	628
10.3.6.16	Bars	628
10.3.6.17	Meter	630
10.3.6.18	HAL_Graph	631
10.3.6.19	Grmlin tool path preview for .ngc files	631
10.3.6.20	HAL_Offset	634
10.3.6.21	DRO widget	634
10.3.6.22	Combi_DRO widget	635
10.3.6.23	ConView (File Select)	639
10.3.6.24	Calculator widget	642
10.3.6.25	Tooleditor widget	642

10.3.6.2	Offsetpage	644
10.3.6.2	HAL_sourceview widget	646
10.3.6.2	MDI history	647
10.3.6.2	Animated function diagrams: HAL widgets in a bitmap	648
10.3.7	Action Widgets Reference	648
10.3.7.1	VCP Action widgets	649
10.3.7.2	VCP Action python	649
10.3.7.3	VCP ToggleAction widgets	650
10.3.7.4	The Action_MDI Toggle and Action_MDI widgets	650
10.3.7.5	A simple example: Execute MDI command on button press	651
10.3.7.6	Parameter passing with Action_MDI and ToggleAction_MDI widgets	651
10.3.7.7	An advanced example: Feeding parameters to an O-word subroutine	652
10.3.7.8	Preparing for an MDI Action, and cleaning up afterwards	652
10.3.7.9	Using the LinuxCNC Stat object to deal with status changes	653
10.3.8	GladeVCP Programming	654
10.3.8.1	User Defined Actions	654
10.3.8.2	Core Library	654
10.3.8.3	An example: adding custom user callbacks in Python	655
10.3.8.4	HAL value change events	655
10.3.8.5	Programming model	655
10.3.8.6	Initialization sequence	657
10.3.8.7	Multiple callbacks with the same name	658
10.3.8.8	The GladeVCP -U <useropts> flag	658
10.3.8.9	Persistent variables in GladeVCP	658
10.3.8.10	Using persistent variables	658
10.3.8.11	Saving the state on GladeVCP shutdown	659
10.3.8.12	Saving state when Ctrl-C is pressed	660
10.3.8.13	Hand-editing INI (.ini) files	660
10.3.8.14	Adding HAL pins	660
10.3.8.15	Adding timers	661
10.3.8.16	Setting HAL widget properties programmatically	661
10.3.8.17	Examples, and rolling your own GladeVCP application	662
10.3.9	FAQ	662
10.3.10	Troubleshooting	663
10.3.11	Implementation note: Key handling in AXIS	663
10.3.12	Adding Custom Widgets	663
10.3.13	Auxiliary GladeVCP Applications	663
10.4	Qtvc	664
10.4.1	Overview	672

10.4.1.1Qtvcp Widgets	672
10.4.1.2INI Settings	672
10.4.1.3QTDesigner UI File	673
10.4.1.4Handler Files	673
10.4.1.5Libraries modules	673
10.4.1.6Themes	673
10.4.1.7Local Files	673
10.4.1.8Modifying Stock Screens	674
10.4.2VCP Panels	674
10.4.2.1Builtin panels	674
10.4.3Build a simple clean-sheet custom screen	676
10.4.3.1Overview	676
10.4.3.2Get Designer to include linuxcnc widgets	677
10.4.3.3build the screen .ui file	677
10.4.3.4Handler file	680
10.4.3.5INI	680
10.4.3.6HAL	680
10.4.4Handler file in detail	680
10.4.4.1Overview	680
10.4.4.2IMPORT SECTION	684
10.4.4.3INSTANTIATE LIBRARIES SECTION	684
10.4.4.4HANDLER CLASS section	684
10.4.4.5INITIALIZE section	684
10.4.4.6SPECIAL FUNCTIONS section	684
10.4.4.7STATUS CALLBACKS section	685
10.4.4.8CALLBACKS FROM FORM section	685
10.4.4.9GENERAL FUNCTIONS section	685
10.4.4.10KEY BINDING section	685
10.4.4.11CLOSING EVENT section	685
10.4.5Connecting widgets to python code	685
10.4.5.1Overview	686
10.4.5.2Using Designer to add slots	686
10.4.5.3Handler file changes	687
10.4.5.4More Information	687
10.4.6QtVCP Virtual Control Panels	687
10.4.6.1Builtin panels	687
10.4.6.2Custom Panels	698
10.4.6.3More Information	699
10.4.7QtVCP Widgets	699

10.4.7.1	HAL Only Widgets	699
10.4.7.2	Machine Controller Widgets	708
10.4.7.3	StatusLabel Widget	719
10.4.7.4	Dialog Widgets	739
10.4.7.5	Other	747
10.4.7.6	Import only Widgets	750
10.4.8	QTVcp Libraries modules	751
10.4.8.1	STATUS	751
10.4.8.2	INFO	752
10.4.8.3	Action	754
10.4.8.4	TOOL	756
10.4.8.5	Path	756
10.4.8.6	VCPWindow	757
10.4.8.7	aux_program_loader	758
10.4.8.8	Keybindings	758
10.4.8.9	Messages	759
10.4.8.10	otify	760
10.4.8.11	preferences	760
10.4.8.12	audio player	761
10.4.8.13	irtual keyboard	762
10.4.8.14	TOOLBAR ACTIONS	762
10.4.8.15	QT Vismach Machine Graphics library	763
10.4.9	QT Vismach	765
10.4.9.1	Hierarchy of Machine Design	767
10.4.9.2	Start the script	767
10.4.9.3	HAL pins	768
10.4.9.4	Creating Parts	768
10.4.9.5	Orientating Model Parts	769
10.4.9.6	Animating Parts	769
10.4.9.7	Assembling the model	770
10.4.9.8	Other functions	771
10.4.9.9	Basic structure of a QT Vismach script	771
10.4.9.10	Builtin Vismach Sample Panels	772
10.4.10	QtVCP Building Custom Widgets	773
10.4.10.0	Overview	773
10.4.10.0	Custom Hal Widgets	773
10.4.10.0	Custom Controller Widgets using STATUS	774
10.4.10.0	Custom Controller Widgets with actions	777
10.4.10.0	Stylesheet Property Changes Based On Events	779

10.4.10.	Use Stylesheets To Change Custom Widget Properties	780
10.4.10.	Widget Plugins	780
10.4.10.	QtVCP Handler file code snippets	784
10.4.11.	Preference file loading/saving	784
10.4.11.	Use QSettings to read/save variable	785
10.4.11.	Add a basic style editor	785
10.4.11.	Request Dialog Entry	786
10.4.11.	Speak a Startup Greeting	787
10.4.11.	ToolBar Functions.	787
10.4.11.	Add HAL Pins that call functions	788
10.4.11.	Add a special Max Velocity Slider based on percent	788
10.4.11.	Toggle Continuous Jog On and Off	789
10.4.11.	Class Patch the file manager widget	790
10.4.11.	Adding widgets Programmatically	791
10.4.11.	Update/Read Objects Periodically	793
10.4.11.	External control with ZMQ messaging reading	794
10.4.11.	External control with ZMQ messaging writing	795
10.4.11.	Sending messages to status bar or desktop notify dialogs	796
10.4.11.	Catch Focus Changes	797
10.4.10.	Qtvc Development	798
10.4.12.	Overview	798
10.4.12.	Builtin locations	798
10.4.12.	QtVCP startup to shutdown	798
10.4.12.	Path information	799
10.4.12.	Idiosyncrasies	799
11	User Interfaces	801
11.1	Panelui	801
11.1.1	Introduction	801
11.1.2	Loading Commands	801
11.1.3	panelui.ini file reference	802
11.1.4	Internal Command reference	804
11.1.5	ZMQ Messages	806
11.1.6	Handler File Extension	807
11.2	HAL User Interface	808
11.2.1	Introduction	808
11.2.2	MDI	808
11.2.3	Example Configuration	809
11.2.4	Halui Pin Reference	809

11.2.4.1	Abort	809
11.2.4.2	E-Stop	809
11.2.4.3	Feed Override	809
11.2.4.4	Mist	810
11.2.4.5	Flood	810
11.2.4.6	Homing	810
11.2.4.7	Lube	810
11.2.4.8	Machine	810
11.2.4.9	Max Velocity	811
11.2.4.1	MDI	811
11.2.4.1	Joint	811
11.2.4.1	Joint Jogging	812
11.2.4.1	Axis	813
11.2.4.1	Axis Jogging	813
11.2.4.1	Mode	814
11.2.4.1	Program	814
11.2.4.1	Rapid Override	814
11.2.4.1	Spindle Override	815
11.2.4.1	Spindle	815
11.2.4.2	Tool	816
11.3	Halui Examples	816
11.3.1	Remote Start	816
11.3.2	Pause & Resume	817
11.4	Vismach	818
11.4.1	Start the script	820
11.4.2	Create the HAL pins.	820
11.4.3	Creating Parts	820
11.4.4	Moving Parts	821
11.4.5	Animating Parts	821
11.4.6	Assembling the model.	822
11.4.7	Other functions	823
11.4.8	Basic structure of a Vismach script.	823
11.5	Python Interface	824
11.5.1	The linuxcnc Python module	824
11.5.2	Usage Patterns for the LinuxCNC NML interface	824
11.5.3	Reading LinuxCNC status	825
11.5.3.1	linuxcnc.stat attributes	825
11.5.3.2	The axis dictionary	830
11.5.3.3	The joint dictionary	830

11.5.4	The spindle dictionary	832
11.5.5	Preparing to send commands	832
11.5.6	Sending commands through <code>linuxcnc.command</code>	833
11.5.6.1	<code>linuxcnc.command</code> attributes	834
11.5.6.2	<code>linuxcnc.command</code> methods:	834
11.5.7	Reading the error channel	837
11.5.8	Reading INI file values	837
11.5.9	The <code>linuxcnc.positionlogger</code> type	838
11.5.9.1	members	838
11.5.9.2	methods	838
11.6	GStat	839
11.6.1	Intro	839
11.6.2	Overview	839
11.6.3	Sample GStat Code	839
11.6.3.1	Sample HAL component code pattern	839
11.6.3.2	GladeVCP python extension code pattern	840
11.6.3.3	QTVCP python extension code pattern	841
11.6.4	Messages	842
11.6.5	Functions	847
11.6.6	Known Issues	850
12	Drivers	851
12.1	Parallel Port Driver	851
12.1.1	Loading	851
12.1.2	PCI Port Address	854
12.1.3	Pins	855
12.1.4	Parameters	855
12.1.5	Functions	855
12.1.6	Common problems	856
12.1.7	Using DoubleStep	856
12.1.8	<code>probe_parport</code>	856
12.1.8.1	Installing <code>probe_parport</code>	856
12.2	AX5214H Driver	857
12.2.1	Installing	857
12.2.2	Pins	857
12.2.3	Parameters	857
12.2.4	Functions	857
12.3	GS2 VFD Driver	858
12.3.1	Command Line Options	858

12.3.2	Pins	859
12.3.3	Parameters	859
12.4	Mesa HostMot2 Driver	860
12.4.1	Introduction	860
12.4.2	Firmware Binaries	860
12.4.3	Installing Firmware	860
12.4.4	Loading HostMot2	861
12.4.5	Watchdog	861
12.4.5.1	Pins:	861
12.4.5.2	Parameters:	861
12.4.6	HostMot2 Functions	861
12.4.7	Pinouts	862
12.4.8	PIN Files	863
12.4.9	Firmware	863
12.4.10	HAL Pins	863
12.4.11	Configurations	864
12.4.12	GPIO	866
12.4.12.1	Pins	866
12.4.12.2	Parameters	867
12.4.13	StepGen	867
12.4.13.1	Pins	867
12.4.13.2	Parameters	868
12.4.13.3	Output Parameters	868
12.4.14	PWMGen	868
12.4.14.1	Pins	868
12.4.14.2	Parameters	869
12.4.14.3	Output Parameters	869
12.4.15	Encoder	869
12.4.15.1	Pins	870
12.4.15.2	Parameters	870
12.4.16	i25 Configuration	871
12.4.16.1	Firmware	871
12.4.16.2	Configuration	871
12.4.16.3	SSERIAL Configuration	871
12.4.16.4	i77 Limits	871
12.4.17	Example Configurations	872
12.5	Motenc Driver	872
12.5.1	Pins	872
12.5.2	Parameters	873

12.5.3Functions	873
12.6MB2HAL	874
12.6.1Introduction	874
12.6.2Usage	874
12.6.3Options	874
12.6.3.1Init Section	874
12.6.3.2Transaction Sections	875
12.6.3.3Error codes	876
12.6.4Example config file	877
12.6.5Pins	881
12.6.5.1fnct_01_read_coils	882
12.6.5.2fnct_02_read_discrete_inputs	882
12.6.5.3fnct_03_read_holding_registers	882
12.6.5.4fnct_04_read_input_registers	882
12.6.5.5fnct_05_write_single_coil	882
12.6.5.6fnct_06_write_single_register	882
12.6.5.7fnct_15_write_multiple_coils	882
12.6.5.8fnct_16_write_multiple_registers	883
12.7Opto22 Driver	883
12.7.1The Adapter Card	883
12.7.2The Driver	883
12.7.3Pins	884
12.7.4Parameters	884
12.7.5FUNCTIONS	884
12.7.6Configuring I/O Ports	884
12.7.7Pin Numbering	885
12.8Pico Drivers	885
12.8.1Command Line Options	886
12.8.2Pins	886
12.8.3Parameters	888
12.8.4Functions	889
12.9Pluto P Driver	889
12.9.1General Info	889
12.9.1.1Requirements	889
12.9.1.2Connectors	889
12.9.1.3Physical Pins	889
12.9.1.4LED	890
12.9.1.5Power	890
12.9.1.6PC interface	890

12.9.1.7	Rebuilding the FPGA firmware	890
12.9.1.8	For more information	890
12.9.2	Pluto Servo	891
12.9.2.1	Pinout	891
12.9.2.2	Input latching and output updating	893
12.9.2.3	HAL Functions, Pins and Parameters	893
12.9.2.4	Compatible driver hardware	893
12.9.3	Pluto Step	893
12.9.3.1	Pinout	894
12.9.3.2	Input latching and output updating	894
12.9.3.3	Step Waveform Timings	895
12.9.3.4	HAL Functions, Pins and Parameters	895
12.10	Powermax Modbus Driver	895
12.10.1	Pins	896
12.10.2	Description	896
12.10.3	Reference:	897
12.11	Servo To Go Driver	897
12.11.1	Installing	897
12.11.2	Pins	897
12.11.3	Parameters	898
12.11.4	Functions	898
12.12	Shuttle	899
12.12.1	Description	899
12.12.2	Setup	899
12.12.3	Pins	899
12.13	General Mechatronics Driver	900
12.13.1	I/O connectors	901
12.13.1.1	Pins	902
12.13.1.2	Parameters	902
12.13.2	Axis connectors	903
12.13.2.1	Axis interface modules	903
12.13.2.2	Encoder	904
12.13.2.3	StepGen module	907
12.13.2.4	Enable and Fault signals	910
12.13.2.5	Axis DAC	910
12.13.3	CAN-bus servo amplifiers	911
12.13.3.1	Pins	912
12.13.3.2	Parameters	912
12.13.4	Watchdog timer	912

12.13.4.Pins	912
12.13.4.Parameters	912
12.13.5.End-, homing- and E-stop switches	913
12.13.5.Pins	914
12.13.5.Parameters	914
12.13.6.Status LEDs	914
12.13.6.CAN	914
12.13.6.RS485	915
12.13.6.BMC	915
12.13.6.Boot	915
12.13.6.Error	915
12.13.7.RS485 I/O expander modules	915
12.13.7.Relay output module	916
12.13.7.Digital input module	917
12.13.7.DAC & ADC module	918
12.13.7.Teach Pendant module	919
12.13.8.Errata	920
12.13.8.GM6-PCI card Errata	920
12.14.VFS11 VFD Driver	920
12.14.Command Line Options	920
12.14.Pins	921
12.14.Parameters	922
12.14.INI file configuration	922
12.14.HAL example	923
12.14.Panel operation	924
12.14.Error Recovery	924
12.14.Configuring the VFS11 VFD for Modbus usage	924
12.14.8.Connecting the Serial Port	924
12.14.8.Modbus setup	925
12.14.Programming Note	925
13 Driver Examples	926
13.1 PCI Parallel Port	926
13.2 Spindle Control	927
13.2.10-10 Volt Spindle Speed	927
13.2.2 PWM Spindle Speed	927
13.2.3 Spindle Enable	927
13.2.4 Spindle Direction	928
13.2.5 Spindle Soft Start	928

13.2.5.1Intro to HAL components limit2 and near	928
13.2.6Spindle Feedback	929
13.2.6.1Spindle Synchronized Motion	929
13.2.6.2Spindle At Speed	930
13.3MPG Pendant	930
13.4GS2 Spindle	932
13.4.1Example	932
14PLC	934
14.1ClassicLadder Introduction	934
14.1.1History	934
14.1.2Introduction	934
14.1.3Example	935
14.1.4Basic Latching On-Off Circuit	936
14.2ClassicLadder Programming	937
14.2.1Ladder Concepts	937
14.2.2Languages	937
14.2.3Components	937
14.2.3.1Files	937
14.2.3.2Realtime Module	938
14.2.3.3Variables	938
14.2.4Loading the ClassicLadder user module	939
14.2.5ClassicLadder GUI	939
14.2.5.1Sections Manager	940
14.2.5.2Section Display	940
14.2.5.3The Variable Windows	942
14.2.5.4Symbol Window	944
14.2.5.5The Editor window	945
14.2.5.6Config Window	946
14.2.6Ladder objects	948
14.2.6.1CONTACTS	948
14.2.6.2IEC TIMERS	948
14.2.6.3TIMERS	949
14.2.6.4MONOSTABLES	949
14.2.6.5COUNTERS	949
14.2.6.6COMPARE	950
14.2.6.7VARIABLE ASSIGNMENT	951
14.2.6.8COILS	953
14.2.7ClassicLadder Variables	954

14.2.8	GRAFCET (State Machine) Programming	955
14.2.9	Modbus	957
14.2.10	MODBUS Settings	960
14.2.10.1	MODBUS Info	961
14.2.10.2	Communication Errors	961
14.2.11	Debugging modbus problems	961
14.2.11.1	Request	963
14.2.11.2	Error response	964
14.2.11.3	Data response	965
14.2.11.4	MODBUS Bugs	966
14.2.12	Setting up ClassicLadder	966
14.2.12.1	Add the Modules	967
14.2.12.2	Adding Ladder Logic	967
14.3	ClassicLadder Examples	972
14.3.1	Wrapping Counter	972
14.3.2	Reject Extra Pulses	973
14.3.3	External E-Stop	974
14.3.4	Timer/Operate Example	977
15	HAL	979
15.1	HAL Introduction	979
15.1.1	HAL System Design	979
15.1.1.1	Part Selection	980
15.1.1.2	Interconnection Design	981
15.1.1.3	Implementation	981
15.1.1.4	Testing	981
15.1.1.5	Summary	981
15.1.2	HAL Concepts	982
15.1.3	HAL components	984
15.1.4	Timing Issues In HAL	984
15.2	HAL Basics	985
15.2.1	HAL Commands	985
15.2.1.1	loadrt	986
15.2.1.2	addf	986
15.2.1.3	loadusr	987
15.2.1.4	net	988
15.2.1.5	setp	989
15.2.1.6	sets	990
15.2.1.7	unlinkp	990

15.2.1.8Obsolete Commands	990
15.2.2HAL Data	991
15.2.2.1Bit	991
15.2.2.2Float	991
15.2.2.3s32	991
15.2.2.4u32	991
15.2.3HAL Files	991
15.2.4HAL Parameter	992
15.2.5Basic Logic Components	992
15.2.5.1and2	992
15.2.5.2not	993
15.2.5.3or2	993
15.2.5.4xor2	994
15.2.6Logic Examples	994
15.2.7Conversion Components	994
15.2.7.1weighted_sum	994
15.3HAL TWOPASS	995
15.3.1TWOPASS	995
15.3.2Post GUI	997
15.3.3Excluding .hal files	997
15.3.4Examples	998
15.4HAL Tutorial	998
15.4.1Introduction	998
15.4.2Halcmd	998
15.4.2.1Notation	999
15.4.2.2Tab-completion	999
15.4.2.3The RTAPI environment	999
15.4.3A Simple Example	999
15.4.3.1Loading a component	999
15.4.3.2Examining the HAL	1000
15.4.3.3Making realtime code run	1001
15.4.3.4Changing Parameters	1002
15.4.3.5Saving the HAL configuration	1003
15.4.3.6Exiting halrun	1004
15.4.3.7Restoring the HAL configuration	1004
15.4.3.8Removing HAL from memory	1004
15.4.4Halmeter	1004
15.4.5Stepgen Example	1007
15.4.5.1Installing the components	1007

15.4.5.2	Connecting pins with signals	1008
15.4.5.3	Setting up realtime execution - threads and functions	1009
15.4.5.4	Setting parameters	1010
15.4.5.5	Run it!	1011
15.4.6	Halscope	1011
15.4.6.1	Hooking up the scope probes	1013
15.4.6.2	Capturing our first waveforms	1016
15.4.6.3	Vertical Adjustments	1017
15.4.6.4	Triggering	1018
15.4.6.5	Horizontal Adjustments	1020
15.4.6.6	More Channels	1021
15.4.6.7	More samples	1022
15.5	HAL General Reference	1023
15.5.1	HAL Entity Names	1023
15.5.2	HAL General Naming Conventions	1023
15.5.3	Hardware Driver Naming Conventions	1023
15.5.3.1	Pins/Parameters names	1024
15.5.3.2	Function Names	1025
15.6	Core Components	1025
15.6.1	Motion	1026
15.6.1.1	Options	1026
15.6.1.2	Pins	1027
15.6.1.3	Parameters	1028
15.6.1.4	Functions	1029
15.6.2	Spindle	1029
15.6.2.1	Pins	1029
15.6.3	Axis and Joint Pins and Parameters	1030
15.6.4	iocontrol	1031
15.6.4.1	Pins	1031
15.6.5	INI settings	1031
15.6.5.1	Pins	1031
15.7	Canonical Device Interfaces	1032
15.7.1	Introduction	1032
15.7.2	Digital Input	1032
15.7.2.1	Pins	1033
15.7.2.2	Parameters	1033
15.7.2.3	Functions	1033
15.7.3	Digital Output	1033
15.7.3.1	Pins	1033

15.7.3.2Parameters	1033
15.7.3.3Functions	1033
15.7.4Analog Input	1033
15.7.4.1Pins	1034
15.7.4.2Parameters	1034
15.7.4.3Functions	1034
15.7.5Analog Output	1034
15.7.5.1Pins	1034
15.7.5.2Parameters	1035
15.7.5.3Functions	1035
15.8HAL Tools	1035
15.8.1Halcmd	1035
15.8.2Halmeter	1036
15.8.3Halshow	1038
15.8.4Halscope	1039
15.8.5Sim Pin	1039
15.8.6Simulate Probe	1040
15.8.7Hal Histogram	1041
15.8.8Halreport	1042
15.9Halshow	1044
15.9.1Starting Halshow	1044
15.9.2HAL Tree Area	1045
15.9.3HAL Show Area	1046
15.9.4Watch Tab Area	1049
15.10HAL Component List	1051
15.10.0Commands and Userspace Components	1051
15.10.1Realtime Components List	1052
15.10.2.0Core LinuxCNC components	1053
15.10.2.1Logic and Bitwise components	1053
15.10.2.2Arithmetic and float-components	1054
15.10.2.3Type conversion	1055
15.10.2.4Hardware Drivers	1056
15.10.2.5Kinematics	1056
15.10.2.6Motor control	1057
15.10.2.7BLDC and 3-phase motor control	1057
15.10.2.8Other Components	1057
15.10.2HAL API calls	1059
15.10.3RTAPI calls	1059
15.11HAL Component Descriptions	1061

15.11.	Stepgen	1061
15.11.1.	Pins	1063
15.11.1.	Parameters	1064
15.11.1.	Step Types	1064
15.11.1.	Functions	1069
15.11.	PWMgen	1069
15.11.2.	Output Types	1070
15.11.2.	Pins	1070
15.11.2.	Parameters	1070
15.11.2.	Functions	1071
15.11.	Encoder	1071
15.11.3.	Pins	1072
15.11.3.	Parameters	1074
15.11.3.	Functions	1074
15.11.	PID	1074
15.11.4.	Pins	1075
15.11.4.	Functions	1077
15.11.	Simulated Encoder	1077
15.11.5.	Pins	1077
15.11.5.	Parameters	1078
15.11.5.	Functions	1078
15.11.	Debounce	1078
15.11.6.	Pins	1078
15.11.6.	Parameters	1079
15.11.6.	Functions	1079
15.11.	Siggen	1079
15.11.7.	Pins	1079
15.11.7.	Parameters	1080
15.11.7.	Functions	1080
15.11.	Bit5	1080
15.1	HAL Examples	1081
15.12.	Connecting Two Outputs	1082
15.12.	Manual Toolchange	1082
15.12.	Compute Velocity	1083
15.12.	Soft Start Details	1084
15.12.	Stand Alone HAL	1086
15.1	HAL Component Generator	1087
15.13.	Introduction	1087
15.13.	Installing	1088

15.13.Using a Component	1088
15.13.Definitions	1088
15.13.Instance creation	1088
15.13.Implicit Parameters	1089
15.13.Syntax	1089
15.13.7.HAL functions	1091
15.13.7.Options	1091
15.13.7.License and Authorship	1092
15.13.7.Per-instance data storage	1092
15.13.7.Comments	1093
15.13.Restrictions	1093
15.13.Convenience Macros	1093
15.13.Components with one function	1094
15.13.Component Personality	1094
15.13.Compiling	1095
15.13.Compiling realtime components outside the source tree	1095
15.13.Compiling userspace components outside the source tree	1095
15.13.Examples	1096
15.13.15.constant	1096
15.13.15.sin	1096
15.13.15.cos	1096
15.13.15.out8	1096
15.13.15.hal_loop	1097
15.13.15.afraidemo	1097
15.13.15.rand	1098
15.13.15.logic	1098
15.13.15.General Functions	1099
15.13.Command Line Usage	1099
15.14.HALTCL Files	1100
15.14.Compatibility	1100
15.14.Haltcl Commands	1100
15.14.Haltcl INI-file variables	1100
15.14.Converting .hal files to .tcl files	1102
15.14.Haltcl Notes	1102
15.14.Haltcl Examples	1102
15.14.Haltcl Interactive	1103
15.14.Haltcl Distribution Examples (sim)	1103
15.15.Creating Userspace Python Components	1103
15.15.Basic usage	1103
15.15.Userspace components and delays	1104

15.15. Create pins and parameters	1104
15.15.3. Changing the prefix	1105
15.15. Reading and writing pins and parameters	1105
15.15.4. Driving output (HAL_OUT) pins	1105
15.15.4. Driving bidirectional (HAL_IO) pins	1105
15.15. Exiting	1105
15.15. Helpful Functions	1105
15.15.6. ready	1105
15.15.6. unready	1106
15.15.6. component_exists	1106
15.15.6. component_is_ready	1106
15.15.6. get_msg_level	1106
15.15.6. set_msg_level	1106
15.15.6. connect	1106
15.15.6. disconnect	1106
15.15.6. get_value	1106
15.15.6. get_info_pins()	1107
15.15.6. get_info_signals()	1107
15.15.6. get_info_params()	1107
15.15.6. new_sig	1107
15.15.6. pin_has_writer	1107
15.15.6. get_name	1107
15.15.6. get_type	1108
15.15.6. get_dir	1108
15.15.6. get	1108
15.15.6. set	1108
15.15.6. setpin	1108
15.15.6. sampler_base	1108
15.15.6. stream_base	1108
15.15.6. stream	1108
15.15.6. set_p	1109
15.15. Constants	1109
15.15. System Information	1109
15.15. Use with hal_glib in GladeVCP Handler	1110
15.15. Use with hal_glib in QtVCP Handler	1110
15.15. Project ideas	1111

V Advanced Topics	1112
16 Kinematics	1113
16.1 Introduction	1113
16.1.1 Joints vs Axes	1113
16.2 Trivial Kinematics	1113
16.3 Non-trivial kinematics	1115
16.3.1 Forward transformation	1116
16.3.2 Inverse transformation	1117
16.4 Implementation details	1117
16.4.1 Kinematics module using the userkins.comp template	1118
17 Setting up "modified" Denavit-Hartenberg (DH) parameters for <i>genserkins</i>	1119
17.1 Prelude	1119
17.2 General	1119
17.3 Modified DH-Parameters	1120
17.4 Modified DH-Parameters as used in <i>genserkins</i>	1120
17.5 Numbering of joints and parameters	1121
17.6 How to start	1121
17.7 Special cases	1121
17.8 Detailed Example (RV-6SL)	1121
17.9 Credits	1140
18 5-Axis Kinematics	1141
18.1 Introduction	1141
18.2 5-Axis Machine Tool Configurations	1141
18.3 Tool Orientation and Location	1141
18.4 Translation and Rotation Matrices	1142
18.5 Table Rotary/Tilting 5-Axis Configurations	1143
18.5.1 Transformations for a xyzac-trt machine tool with work offsets	1145
18.5.1.1 Forward transformation	1146
18.5.1.2 Inverse Transformation	1148
18.5.2 Transformations for a xyzac-trt machine with rotary axis offsets	1149
18.5.2.1 Forward Transformation	1150
18.5.2.2 Inverse Transformation	1151
18.5.3 Transformations for a xyzbc-trt machine with rotary axis offsets	1152
18.5.3.1 Forward Transformation	1153
18.5.3.2 Inverse Transformation	1154
18.6 Table Rotary/Tilting Examples	1155
18.6.1 Vismach Simulation Models	1155
18.6.2 Tool-Length Compensation	1155
18.7 Custom Kinematics Components	1156
18.8 Figures	1157
18.9 REFERENCES	1159

19 Switchable Kinematics (switchkins)	1160
19.1 Introduction	1160
19.2 Switchable Kinematic Modules	1160
19.2.1 Identity letter assignments	1161
19.2.2 Backwards compatibility	1161
19.3 HAL Pins	1161
19.3.1 HAL Pin Summary	1162
19.4 Usage	1162
19.4.1 HAL Connections	1162
19.4.2 G-/M-code commands	1162
19.4.3 INI file limit settings	1163
19.4.4 Offset considerations	1164
19.5 Simulation configs	1165
19.6 User kinematics provisions	1165
19.7 Warnings	1165
19.8 Code Notes	1166
20 PID Tuning	1167
20.1 PID Controller	1167
20.1.1 Control loop basics	1167
20.1.2 Theory	1168
20.1.3 Loop Tuning	1168
21 External Axis Offsets	1170
21.1 INI File Settings	1170
21.2 HAL Pins	1170
21.2.1 Per-Axis Motion HAL Pins	1170
21.2.2 Other Motion HAL Pins	1171
21.3 Usage	1171
21.3.1 Offset Computation	1171
21.3.2 Machine-off/Machine-on	1171
21.3.3 Soft Limits	1172
21.3.4 Notes	1172
21.3.5 Warning	1173
21.4 Related HAL Components	1173
21.4.1 eoffset_per_angle.comp	1173
21.5 Testing	1173
21.6 Examples	1174
21.6.1 eoffsets.ini	1174
21.6.2 jwp_z.ini	1174
21.6.3 dynamic_offsets.ini	1174
21.6.4 opa.ini (eoffset_per_angle)	1175

22 Tool Database Interface	1176
22.1 Interface	1176
22.1.1 INI file Settings	1176
22.1.2 db_program operation (v2.0)	1176
22.1.3 Usage	1177
22.1.3.1 Debug Environmental Variables	1178
22.1.4 Example program	1178
22.1.5 Python tooldb module	1179
22.2 Simulation configs	1179
22.2.1 Notes	1180
23 Stand Alone Interpreter	1181
23.1 Usage	1181
23.2 Example	1181
VI Glossary	1183
VII Legal Section	1189
24 Copyright Terms	1191
25 GNU Free Documentation License	1192
26 Index	1197

Part I

Contents

Part II

About LinuxCNC

Chapter 1

Introduction



This handbook is a work in progress. If you are able to help with writing, editing, or graphic preparation please contact any member of the writing team or join and send an email to emc-users@lists.sourceforge.net

Copyright © 2000-2020 LinuxCNC.org

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.1 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

If you do not find the license you may order a copy from:

Free Software Foundation, Inc.
51 Franklin Street
Fifth Floor
Boston, MA 02110-1301 USA.

(The English language version is authoritative)

LINUX® is the registered trademark of Linus Torvalds in the U.S. and other countries. The registered trademark Linux® is used pursuant to a sublicense from LMI, the exclusive licensee of Linus Torvalds, owner of the mark on a world-wide basis.

The LinuxCNC project is not affiliated with Debian®. *Debian* is a registered trademark owned by Software in the Public Interest, Inc.

The LinuxCNC project is not affiliated with UBUNTU®. *UBUNTU* is a registered trademark owned by Canonical Limited.

Chapter 2

LinuxCNC History

Chapter 3

Origin

EMC (the Enhanced Machine Controller) was created by [NIST](#), the National Institute of Standards and Technology, which is an agency of the Commerce Department of the United States government.

NIST first became interested in writing a motion control package as a test platform for concepts and standards. Early sponsorship from General Motors resulted in an adaptation of the fledgling version of EMC using PMAC intelligent control boards running under a "real time" version of Windows NT and controlling a large milling machine.

As is required of all *work product* of US federal government employees, the resulting software and the report about it are required to be in the public domain and a report about it was duly published, including on the Internet. It was there that Matt Shaver discovered EMC. He contacted NIST and entered into discussions with Fred Proctor about adapting the code for use in controlling less expensive hardware to be used for upgrades and replacements of CNC controls that were obsolete or just plain dead. NIST was intrigued because they too wanted something less expensive. In order to launch a cooperative effort, a formal agreement was created which guaranteed that the resulting code and design would remain in the public domain.

Early considerations focused on replacing the expensive and temperamental "real time" Windows NT system. It was proposed that a relatively new (at the time) real time extension of the Linux operating system be tried. This idea was pursued with success. Next up was the issue of the expensive intelligent motion control boards. By this time the processing power of a PC was considered great enough to directly take control of the motion routines. A quick search of available hardware resulted in the selection of a [Servo-To-Go](#) interface board as the first platform for letting the PC directly control the motors. Software for trajectory planning and PID loop control was added to the existing user interface and RS274 interpreter. Matt successfully used this version to upgrade a couple of machines with dead controls and this became the EMC system that first caught the attention of the outside world. Mention of EMC on the [rec.crafts.metalworking](#) USENET newsgroup resulted in early adopters like [Jon Elson](#) building systems to take advantage of EMC.

NIST set up a mailing list for people interested in EMC. As time went on, others outside NIST became interested in improving EMC. Many people requested or coded small improvements to the code. Ray Henry wanted to refine the user interface. Since Ray was reluctant to try tampering with the C code in which the user interface was written, a simpler method was sought. Fred Proctor of NIST suggested a scripting language and wrote code to interface the Tcl/Tk scripting language to the internal NML communications of EMC. With this tool Ray went on to write a Tcl/Tk program that became the predominant user interface for EMC at the time.

For NIST's perspective, see this [paper](#) written by William Shackleford and Frederick Proctor, describing the history of EMC and its transition to open source.

By this time interest in EMC as beginning to pick up substantially. As more and more people attempted installation of EMC, the difficulty of patching a Linux kernel with the real time extensions and of compiling the EMC code became glaringly obvious. Many attempts to document the process and

write scripts were attempted, some with moderate success. The problem of matching the correct version of the patches and compilers with the selected version of Linux kept cropping up. Paul Corner came to the rescue with the BDI (brain dead install) which was a CD from which a complete working system (Linux, patches, and EMC) could be installed. The BDI approach opened the world of EMC to a much larger user community. As this community continued to grow, the EMC mailing list and code archives were moved to [SourceForge](#) and the LinuxCNC web site was established.

With a larger community of users participating, EMC became a major focus of interest at the on-going CNC exhibits at NAMES and NAMES became the annual meeting event for EMC. For the first couple of years, the meetings just happened because the interested parties were at NAMES. In 2003 the EMC user community had its first announced public meeting. It was held the Monday after NAMES in the lobby of the arena where the NAMES show was held. Organization was loose, but the idea of a hardware abstraction layer (HAL) was born and the movement to restructure the code for ease of development (EMC2) was proposed.

3.1 Name Change

In the spring of 2011, the LinuxCNC Board of Directors was contacted by a law firm representing EMC Corporation (www.emc.com) about the use of "EMC" and "EMC2" to identify the software offered on linuxcnc.org. EMC Corporation has registered various trademarks relating to EMC and EMC² (EMC with superscripted numeral two).

After a number of conversations with the representative of EMC Corporation, the final result is that, starting with the next major release of the software, linuxcnc.org will stop identifying the software using "emc" or "EMC", or those terms followed by digits. To the extent that the LinuxCNC Board of Directors controls the names used to identify the software offered on linuxcnc.org, the board has agreed to this.

As a result, it was necessary to choose a new name for the software. Of the options the board considered, there was consensus that "LinuxCNC" is the best option, as this has been our website's name for years.

In preparation for the new name, we have received a sub-license of the LINUX® trademark from the Linux Foundation (www.linuxfoundation.org), protecting our use of the LinuxCNC name. (LINUX® is the registered trademark of Linus Torvalds in the U.S. and other countries.)

The rebranding effort included the linuxcnc.org website, the IRC channels, and versions of the software and documentation since version 2.5.0.

3.2 Additional Info

NIST published a paper describing the [RS274NGC](#) language and the abstract machining center it controls, as well as an early implementation of EMC. The paper is also available at <http://linuxcnc.org/files/RS274NGCv3.pdf>.

NIST also published a paper on the history of EMC and its transition to [open source](#). The paper is also available at <http://linuxcnc.org/files/Use-of-Open-Source-Distribution-for-a-Machine-Tool-Controller.pdf>

Part III

Using LinuxCNC

Chapter 4

General Info

4.1 User Foreword

LinuxCNC is modular and flexible. These attributes lead many to see it as a confusing jumble of little things and wonder why it is the way it is. This page attempts to answer that question before you get into the thick of things.

LinuxCNC started at the National Institute of Standards and Technology in the USA. It grew up using Unix as its operating system. Unix made it different. Among early Unix developers there grew a set of code writing ideas that some call the Unix way. These early LinuxCNC authors followed those ways.

Eric S. Raymond, in his book *The Art of Unix Programming*, summarizes the Unix philosophy as the widely-used engineering philosophy, "Keep it Simple, Stupid" (KISS Principle). He then describes how he believes this overall philosophy is applied as a cultural Unix norm, although unsurprisingly it is not difficult to find severe violations of most of the following in actual Unix practice:

- Rule of Modularity: Write simple parts connected by clean interfaces.
- Rule of Clarity: Clarity is better than cleverness.
- Rule of Composition: Design programs to be connected to other programs.
- Rule of Separation: Separate policy from mechanism; separate interfaces from engines.¹

Mr. Raymond offered several more rules but these four describe essential characteristics of the LinuxCNC motion control system.

The **Modularity** rule is critical. Throughout these handbooks you will find talk of the interpreter or task planner or motion or HAL. Each of these is a module or collection of modules. It's modularity that allows you to connect together just the parts you need to run your machine.

The **Clarity** rule is essential. LinuxCNC is a work in progress — it is not finished nor will it ever be. It is complete enough to run most of the machines we want it to run. Much of that progress is achieved because many users and code developers are able to look at the work of others and build on what they have done.

The **Composition** rule allows us to build a predictable control system from the many modules available by making them connectable. We achieve connectability by setting up standard interfaces to sets of modules and following those standards.

The **Separation** rule requires that we make distinct parts that do little things. By separating functions debugging is much easier and replacement modules can be dropped into the system and comparisons easily made.

¹Found at http://en.wikipedia.org/wiki/Unix_philosophy, 07/06/2008

What does the Unix way mean for you as a user of LinuxCNC. It means that you are able to make choices about how you will use the system. Many of these choices are a part of machine integration, but many also affect the way you will use your machine. As you read you will find many places where you will need to make comparisons. Eventually you will make choices, "I'll use this interface rather than that" or, "I'll write part offsets this way rather than that way." Throughout these handbooks we describe the range of abilities currently available.

As you begin your journey into using LinuxCNC we offer two cautionary notes:²

- Paraphrasing the words of Doug Gwyn on UNIX: "LinuxCNC was not designed to stop its users from doing stupid things, as that would also stop them from doing clever things."
- Likewise the words of Steven King: "LinuxCNC is user-friendly. It just isn't promiscuous about which users it's friendly with."

A series of videos on YouTube provide plenty of evidence a transition to LinuxCNC is possible no matter what your regular computer operating system may be. That said, with the advent of additive manufacturing like 3D printing there is an increasing interest by the broader IT community in CNC machining and it should be possible to find someone with complementary skills/equipment near to you to jointly overcome the initial hurdles.

4.2 LinuxCNC User Introduction

4.2.1 Introduction

This document is focused on the use of LinuxCNC, it is intended for readers who have already installed and configured it. Some information on installation is given in the following chapters. The complete documentation on installation and configuration can be found in the integrator's manual.

4.2.2 How LinuxCNC Works

LinuxCNC is a suite of highly-customisable applications for the control of a Computer Numerically Controlled (CNC) mills and lathes, 3D printers, robots, laser cutters, plasma cutters and other automated devices. It is capable of providing coordinated control of up to 9 axes of movement.

At its heart, LinuxCNC consists of several key components that are integrated together to form one complete system:

- a Graphical User Interface (GUI), which forms the basic interface between the operator, the software and the CNC machine itself;
- the [Hardware Abstraction Layer](#) (HAL), which provides a method of linking all the various internal virtual signals generated and received by LinuxCNC with the outside world; and,
- the high level controllers that coordinate the generation and execution of motion control of the CNC machine, namely the motion controller (EMCMOT), the discrete input/output controller (EMCIO) and the task executor (EMCTASK).

The below illustration is a simple block diagram showing what a typical 3-axis, CNC mill with stepper motors might look like:

²Found at http://en.wikipedia.org/wiki/Unix_philosophy, 07/06/2008

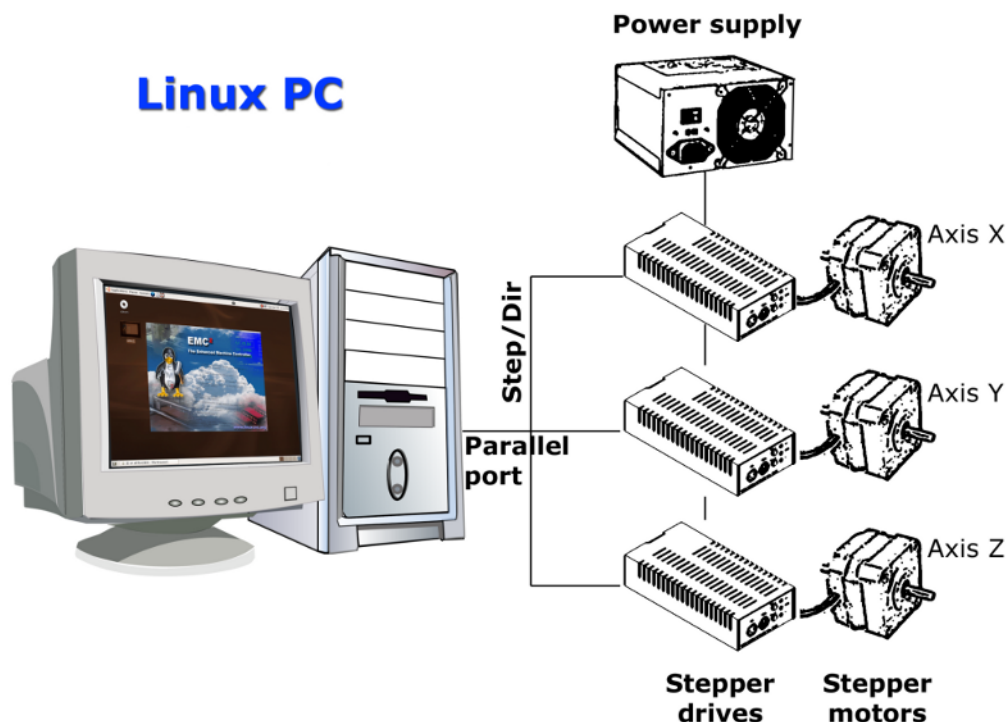


Figure 4.1: Simple LinuxCNC Controlled Machine

A computer running LinuxCNC sends a sequence of pulses via the parallel port to the stepper drives, each of which has one stepper motor connected to it. Each drive receives two independent signals; one signal to command the drive to move its associated stepper motor in a clockwise or anti-clockwise direction, and a second signal that defines the speed at which that stepper motor rotates.

While a stepper motor system under parallel port control is illustrated, a LinuxCNC system can also take advantage of a wide variety of dedicated hardware motion control interfaces for increased speed and I/O capabilities. A full list of interfaces supported by LinuxCNC can be found on the [Supported Hardware](#) page of the Wiki.

In most circumstances, users will create a configuration specific to their mill setup using either the [Stepper Configuration Wizard](#) (for CNC systems operating using the computers' parallel port) or the [Mesa Hardware Wizard](#) (for more advanced systems utilising a Mesa Anything I/O PCI card). Running either wizard will create several folders on the computers' hard drive containing a number of configuration files specific to that CNC machine, and an icon placed on the desktop to allow easy launching of LinuxCNC.

For example, if the Stepper Configuration Wizard was used to create a setup for the 3-axis CNC mill illustrated above entitled *My_CNC*, the folders created by the wizard would typically contain the following files:

- **Folder: My_CNC**

- **My_CNC.ini**

The INI file contains all the basic hardware information regarding the operation of the CNC mill such as the number of steps each stepper motor must turn to complete one full revolution, the maximum rate at which each stepper may operate at, the limits of travel of each axis or the configuration and behaviour of limit switches on each axis.

- **My_CNC.hal**

This HAL file contains information that tells LinuxCNC how to link the internal virtual signals to

physical connections beyond the computer. For example, specifying pin 4 on the parallel port to send out the Z axis step direction signal, or directing LinuxCNC to cease driving the X axis motor when a limit switch is triggered on parallel port pin 13.

- **custom.HAL**

Customisations to the mill configuration beyond the scope of the wizard may be performed by including further links to other virtual points within LinuxCNC in this HAL file. When starting a LinuxCNC session, this file is read and processed before the GUI is loaded. An example may include initiating Modbus communications to the spindle motor so that it is confirmed as operational before the GUI is displayed.

- **custom_postgui.hal**

The custom_postgui HAL file allows further customisation of LinuxCNC, but differs from custom.HAL in that it is processed after the GUI is displayed. For example, after establishing Modbus communications to the spindle motor in custom.hal, LinuxCNC can use the custom_postgui file to link the spindle speed readout from the motor drive to a bargraph displayed on the GUI.

- **postgui_backup.hal**

This is provided as a backup copy of the custom_postgui.hal file to allow the user to quickly restore a previously-working postgui HAL configuration. This is especially useful if the user wants to run the Configuration Wizard again under the same *My_CNC* name in order to modify some parameters of the mill. Saving the mill configuration in the Wizard will overwrite the existing custom_postgui file while leaving the postgui_backup file untouched.

- **tool.tbl**

A tool table file contains a parameterised list of any cutting tools used by the mill. These parameters can include cutter diameter and length, and is used to provide a catalogue of data that tells LinuxCNC how to compensate its motion for different sized tools within a milling operation.

- **Folder: nc_files**

The nc_files folder is provided as a default location to store the G-code programs used to drive the mill. It also includes a number of subfolders with G-code examples.

4.2.3 Graphical User Interfaces

A graphical user interface is the part of the LinuxCNC that the machine tool operator interacts with. LinuxCNC comes with several types of user interfaces which may be chosen from by editing certain fields contained in the [INI file](#):

AXIS

[AXIS](#), the standard keyboard GUI interface. This is also the default GUI launched when a Configuration Wizard is used to create a desktop icon launcher:



Figure 4.2: AXIS, the standard keyboard GUI interface

Touchy

[Touchy](#), a touch screens GUI:



Figure 4.3: Touchy, a touch screen GUI

Gscreen
[Gscreen](#), a user-configurable touch screen GUI:



Figure 4.4: Gscreen, a configurable base touch screen GUI

GMOCCAPY

GMOCCAPY, a touch screen GUI based on Gscreen. GMOCCAPY is also designed to work equally well in applications where a keyboard and mouse are the preferred methods of controlling the GUI:

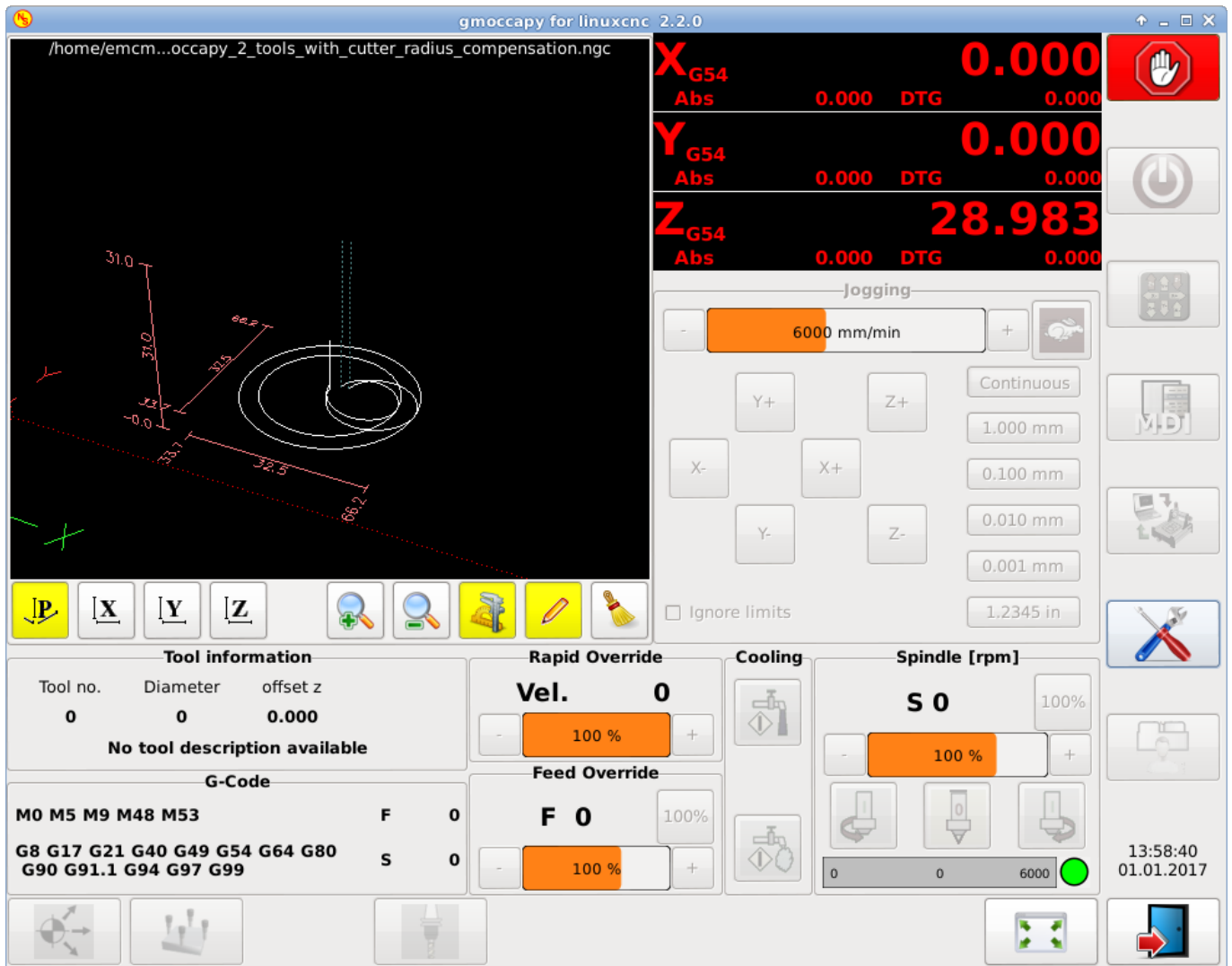


Figure 4.5: GMOCCAPY, a touch screen GUI based on Gscreen

NGCGUI

NGCGUI, a subroutine GUI that provides wizard-style programming of G code. NGCGUI may be run as a standalone program or embedded into another GUI as a series of tabs. The following screen shot shows NGCGUI embedded into Axis:

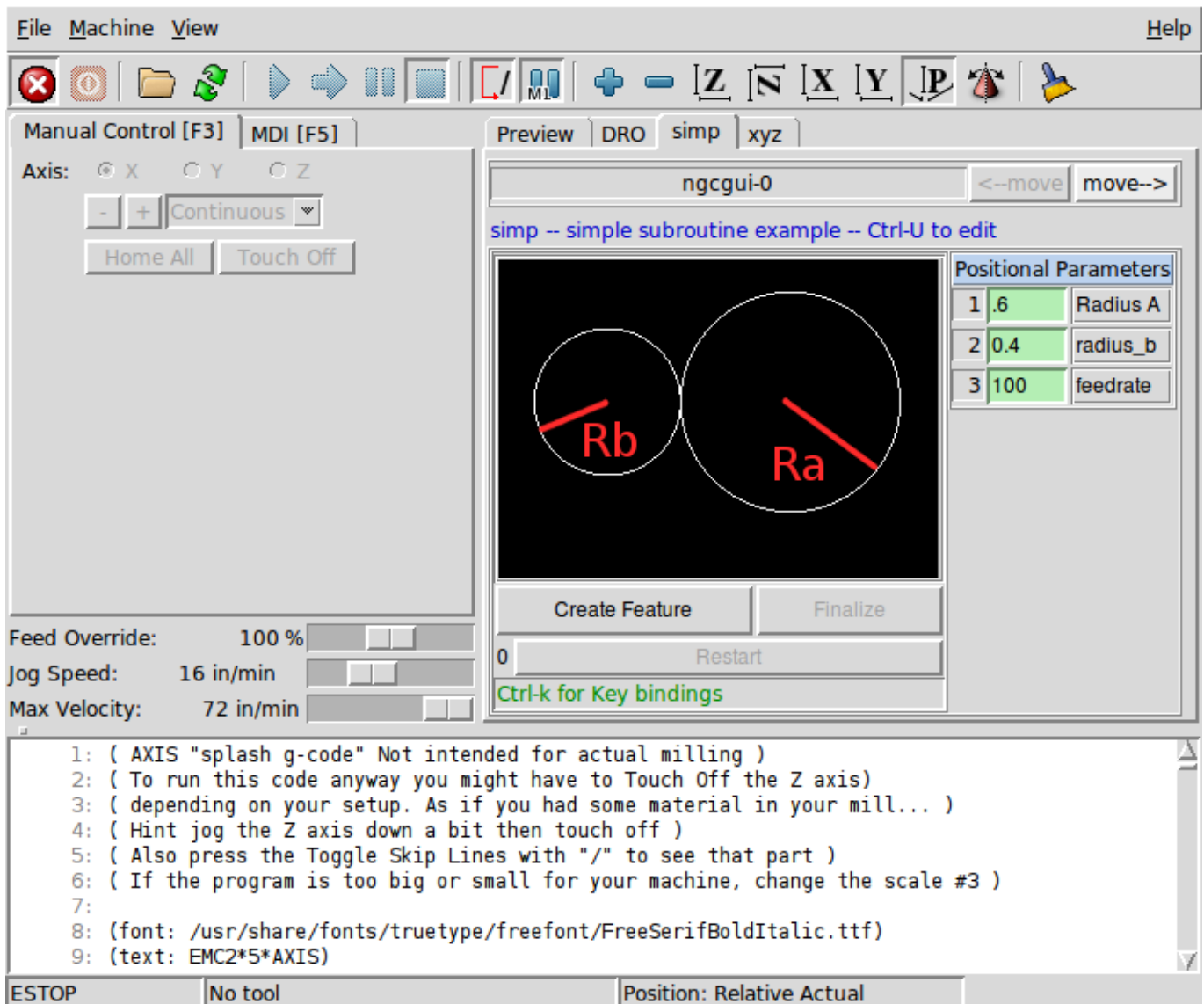


Figure 4.6: NGCGUI, a graphical interface integrated into AXIS

TkLinuxCNC

[TkLinuxCNC](#), another interface based on Tcl/Tk. Once the most popular interface after AXIS.

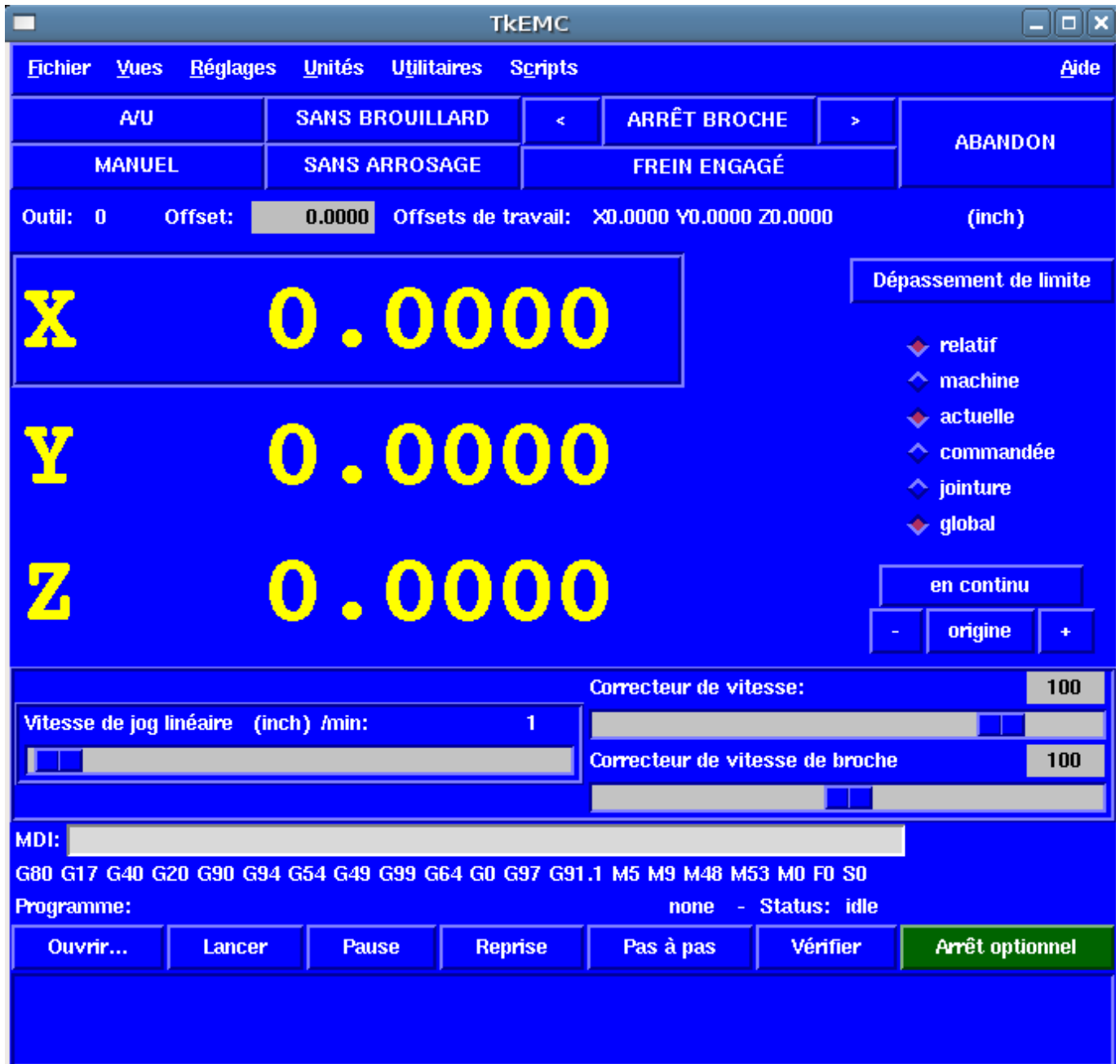


Figure 4.7: TkLinuxCNC graphical interface

Xemc

an X-Window program

halui

A HAL based user interface allowing to control LinuxCNC using buttons and switches

linuxcncrsh

A telnet based user interface allowing to send commands from remote computers.

4.2.4 Virtual Control Panels

As mentioned above, many of LinuxCNC's GUIs may be customized by the user. This may be done to add indicators, readouts, switches or sliders to the basic appearance of one of the GUIs for increased flexibility or functionality. Two styles of Virtual Control Panel are offered in LinuxCNC:

PyVCP

[PyVCP](#), a Python-based virtual control panel that can be added to the Axis GUI. PyVCP only utilises virtual signals contained within the Hardware Abstraction Layer, such as the spindle-at-speed indicator or the Emergency Stop output signal, and has a simple no-frills appearance. This makes it an excellent choice if the user wants to add a Virtual Control Panel with minimal fuss.

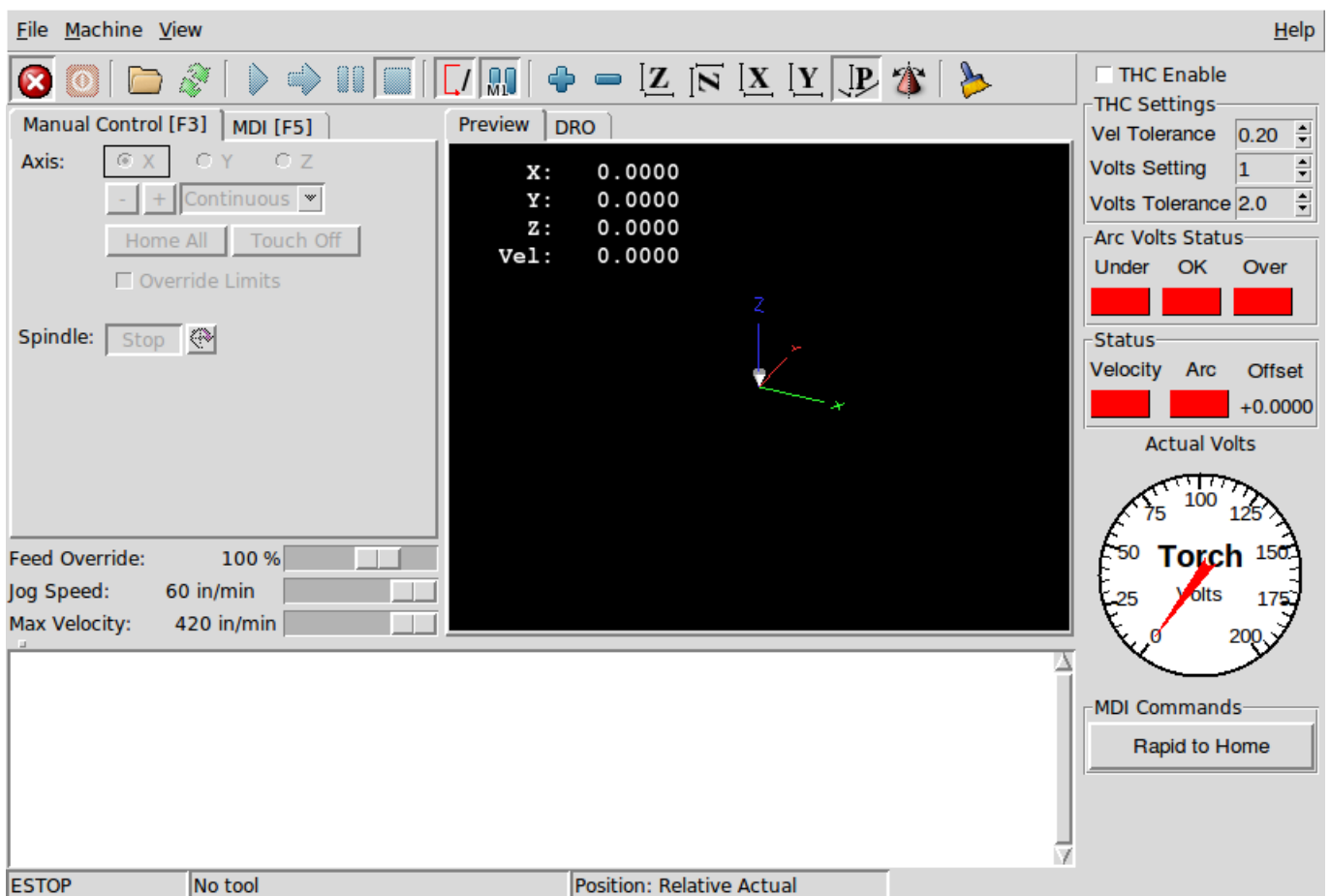


Figure 4.8: PyVCP Example Embedded Into AXIS GUI

GladeVCP

[GladeVCP](#), a Glade-based virtual control panel that can be added to the Axis or Touchy GUIs. GladeVCP has the advantage over PyVCP in that it is not limited to the display or control of HAL virtual signals, but can include other external interfaces outside LinuxCNC such as window or network events. GladeVCP is also more flexible in how it may be configured to appear on the GUI:



Figure 4.9: GladeVCP Example Embedded Into AXIS GUI

4.2.5 Languages

LinuxCNC uses translation files to translate LinuxCNC User Interfaces into many languages including French, German, Italian, Finnish, Russian, Romanian, Portuguese and Chinese. Assuming a translation has been created, LinuxCNC will automatically use whatever native language you log in with when starting the Linux operating system. If your language has not been translated, contact a developer on the IRC, the mailing list or the User Forum for assistance.

4.2.6 Think Like a CNC Operator

This manual does not pretend to teach you how to use a lathe or a milling machine. Becoming an experienced operator takes a lot of time and requires a lot of work. An author once said, *We learn by experience, if one possesses it all*. Broken tools, vices attacked and the scars are evidence of the lessons learned. A beautiful finish, tight tolerances and caution during the work are evidence of lessons learned. No machine, none program can replace human experience.

Now that you start working with the LinuxCNC software, you have to put yourself in the shoes of an operator. You must be in the role of someone in charge of a machine. It's a machine that will wait for your commands and then execute the orders that you will give it. In these pages, we will give the explanations which will help you to become a good CNC operator with LinuxCNC.

4.2.7 Modes of Operation

When LinuxCNC is running, there are three different major modes used for inputting commands. These are Manual, Auto, and Manual Data Input (MDI). Changing from one mode to another makes a big difference in the way that the LinuxCNC control behaves. There are specific things that can be done in one mode that cannot be done in another. An operator can home an axis in manual mode but

not in auto or MDI modes. An operator can cause the machine to execute a whole file full of G-codes in the auto mode but not in manual or MDI.

In manual mode, each command is entered separately. In human terms a manual command might be turn on coolant or jog X at 25 inches per minute. These are roughly equivalent to flipping a switch or turning the hand wheel for an axis. These commands are normally handled on one of the graphical interfaces by pressing a button with the mouse or holding down a key on the keyboard. In auto mode, a similar button or key press might be used to load or start the running of a whole program of G-code that is stored in a file. In the MDI mode the operator might type in a block of code and tell the machine to execute it by pressing the <return> or <enter> key on the keyboard.

Some motion control commands are available concurrently and will cause the same changes in motion in all modes. These include Abort, Emergency Stop, and Feed Rate Override. Commands like these should be self explanatory.

The AXIS user interface hides some of the distinctions between Auto and the other modes by making Auto-commands available at most times. It also blurs the distinction between Manual and MDI because some Manual commands like Touch Off are actually implemented by sending MDI commands. It does this by automatically changing to the mode that is needed for the action the user has requested.

4.3 Important User Concepts

This chapter covers important user concepts that should be understood before attempting to run a CNC machine with g code.

4.3.1 Trajectory Control

4.3.1.1 Trajectory Planning

Trajectory planning, in general, is the means by which LinuxCNC follows the path specified by your G-code program, while still operating within the limits of your machinery.

A G-code program can never be fully obeyed. For example, imagine you specify as a single-line program the following move:

```
G1 X1 F10 (G1 is linear move, X1 is the destination, F10 is the speed)
```

In reality, the whole move can't be made at F10, since the machine must accelerate from a stop, move toward X=1, and then decelerate to stop again. Sometimes part of the move is done at F10, but for many moves, especially short ones, the specified feed rate is never reached at all. Having short moves in your G-code can cause your machine to slow down and speed up for the longer moves if the *naive cam detector* is not employed with G64 Pn.

The basic acceleration and deceleration described above is not complex and there is no compromise to be made. In the INI file the specified machine constraints such as maximum axis velocity and axis acceleration must be obeyed by the trajectory planner.

For more information on the Trajectory Planner ini options see the [Trajectory Section](#) in the INI chapter.

4.3.1.2 Path Following

A less straightforward problem is that of path following. When you program a corner in G Code, the trajectory planner can do several things, all of which are right in some cases:

- it can decelerate to a stop exactly at the coordinates of the corner, and then accelerate in the new direction.

- It can also do what is called blending, which is to keep the feed rate up while going through the corner, making it necessary to round the corner off in order to obey machine constraints.

You can see that there is a trade off here: you can slow down to get better path following, or keep the speed up and have worse path following. Depending on the particular cut, the material, the tooling, etc., the programmer may want to compromise differently.

Rapid moves also obey the current trajectory control. With moves long enough to reach maximum velocity on a machine with low acceleration and no path tolerance specified, you can get a fairly round corner.

4.3.1.3 Programming the Planner

The trajectory control commands are as follows:

G61

(Exact Path Mode) G61 visits the programmed point exactly, even though that means it might temporarily come to a complete stop in order to change direction to the next programmed point.

G61.1

(Exact Stop Mode) G61.1 tells the planner to come to an exact stop at every segment's end. The path will be followed exactly but complete feed stops can be destructive for the part or tool, depending on the specifics of the machining.

G64

(Blend Without Tolerance Mode) G64 is the default setting when you start LinuxCNC. G64 is just blending and the naive cam detector is not enabled. G64 and G64 P0 tell the planner to sacrifice path following accuracy in order to keep the feed rate up. This is necessary for some types of material or tooling where exact stops are harmful, and can work great as long as the programmer is careful to keep in mind that the tool's path will be somewhat more curvy than the program specifies. When using G0 (rapid) moves with G64 use caution on clearance moves and allow enough distance to clear obstacles based on the acceleration capabilities of your machine.

G64 P- Q-

(Blend With Tolerance Mode) This enables the *naive cam detector* and enables blending with a tolerance. If you program G64 P0.05, you tell the planner that you want continuous feed, but at programmed corners you want it to slow down enough so that the tool path can stay within 0.05 user units of the programmed path. The exact amount of slowdown depends on the geometry of the programmed corner and the machine constraints, but the only thing the programmer needs to worry about is the tolerance. This gives the programmer complete control over the path following compromise. The blend tolerance can be changed throughout the program as necessary. Beware that a specification of G64 P0 has the same effect as G64 alone (above), which is necessary for backward compatibility for old G Code programs. See the [G64 section](#) of the G code chapter.

Blending without tolerance

The controlled point will touch each specified movement at at least one point. The machine will never move at such a speed that it cannot come to an exact stop at the end of the current movement (or next movement, if you pause when blending has already started). The distance from the end point of the move is as large as it needs to be to keep up the best contouring feed.

Naive CAM Detector

Successive G1 moves that involve only the XYZ axes that deviate less than Q- from a straight line are merged into a single straight line. This merged movement replaces the individual G1 movements for the purposes of blending with tolerance. Between successive movements, the controlled point will pass no more than P- from the actual endpoints of the movements. The controlled point will touch at least one point on each movement. The machine will never move at such a speed that it cannot come to an exact stop at the end of the current movement (or next

movement, if you pause when blending has already started) On G2/3 moves in the G17 (XY) plane when the maximum deviation of an arc from a straight line is less than the G64 Q- tolerance the arc is broken into two lines (from start of arc to midpoint, and from midpoint to end). those lines are then subject to the naive cam algorithm for lines. Thus, line-arc, arc-arc, and arc-line cases as well as line-line benefit from the *naive cam detector*. This improves contouring performance by simplifying the path.

In the following figure the blue line represents the actual machine velocity. The red lines are the acceleration capability of the machine. The horizontal lines below each plot is the planned move. The upper plot shows how the trajectory planner will slow the machine down when short moves are encountered to stay within the limits of the machines acceleration setting to be able to come to an exact stop at the end of the next move. The bottom plot shows the effect of the Naive Cam Detector to combine the moves and do a better job of keeping the velocity as planned.

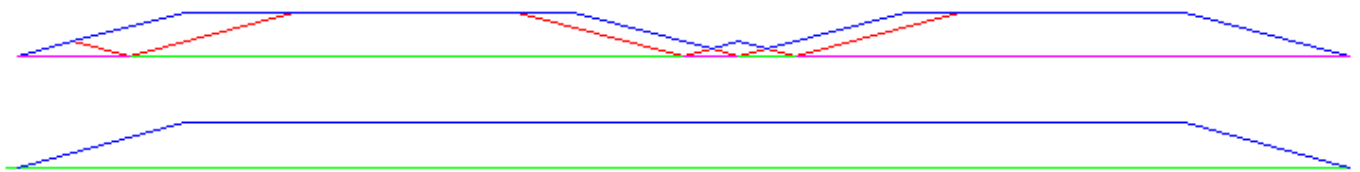


Figure 4.10: Naive CAM Detector

4.3.1.4 Planning Moves

Make sure moves are *long enough* to suit your machine/material. Principally because of the rule that the machine will never move at such a speed that it cannot come to a complete stop at the end of the current movement, there is a minimum movement length that will allow the machine to keep up a requested feed rate with a given acceleration setting.

The acceleration and deceleration phase each use half the ini file MAX_ACCELERATION. In a blend that is an exact reversal, this causes the total axis acceleration to equal the ini file MAX_ACCELERATION. In other cases, the actual machine acceleration is somewhat less than the ini file acceleration

To keep up the feed rate, the move must be longer than the distance it takes to accelerate from 0 to the desired feed rate and then stop again. Using A as $\frac{1}{2}$ the ini file MAX_ACCELERATION and F as the feed rate **in units per second**, the acceleration time is $t_a = F/A$ and the acceleration distance is $d_a = F \cdot t_a / 2$. The deceleration time and distance are the same, making the critical distance $d = d_a + d_d = 2 \cdot d_a = F^2/A$.

For example, for a feed rate of 1 inch per second and an acceleration of **10 inches/sec²**, the critical distance is $1^2/10 = 1/10 = 0.1$ inches.

For a feed rate of 0.5 inch per second, the critical distance is $5^2/100 = 25/100 = 0.025$ inches.

4.3.2 G-code

4.3.2.1 Defaults

When LinuxCNC first starts up many G- and M-codes are loaded by default. The current active G- and M-codes can be viewed on the MDI tab in the *Active G-Codes:* window in the AXIS interface. These G- and M-codes define the behavior of LinuxCNC and it is important that you understand what each one does before running LinuxCNC. The defaults can be changed when running a G-Code file and left in a different state than when you started your LinuxCNC session. The best practice is to set the

defaults needed for the job in the preamble of your G-Code file and not assume that the defaults have not changed. Printing out the G-Code [Quick Reference](#) page can help you remember what each one is.

4.3.2.2 Feed Rate

How the feed rate is applied depends on if an axis involved with the move is a rotary axis. Read and understand the [Feed Rate](#) section if you have a rotary axis or a lathe.

4.3.2.3 Tool Radius Offset

Tool Radius Offset (G41/42) requires that the tool be able to touch somewhere along each programmed move without gouging the two adjacent moves. If that is not possible with the current tool diameter you will get an error. A smaller diameter tool may run without an error on the same path. This means you can program a cutter to pass down a path that is narrower than the cutter without any errors. See the [Cutter Compensation](#) Section for more information.

4.3.3 Homing

After starting LinuxCNC each axis must be homed prior to running a program or running a MDI command.

If your machine does not have home switches a match mark on each axis can aid in homing the machine coordinates to the same place each time.

Once homed your soft limits that are set in the ini file will be used.

If you want to deviate from the default behavior, or want to use the Mini interface you will need to set the option `NO_FORCE_HOMING = 1` in the [TRAJ] section of your ini file. More information on homing can be found in the Integrator Manual.

4.3.4 Tool Changes

There are several options when doing manual tool changes. See the [\[EMCIO\] section](#) for information on configuration of these options. Also see the [G28](#) and [G30](#) section of the G-code chapter.

4.3.5 Coordinate Systems

The Coordinate Systems can be confusing at first. Before running a CNC machine you must understand the basics of the coordinate systems used by LinuxCNC. In depth information on the LinuxCNC Coordinate Systems is in the [Coordinate System](#) Section of this manual.

4.3.5.1 G53 Machine Coordinate

When you home LinuxCNC you set the G53 Machine Coordinate System to 0 for each axis homed.

No other coordinate systems or tool offsets are changed by homing.

The only time you move in the G53 machine coordinate system is when you program a G53 on the same line as a move. Normally you are in the G54 coordinate system.

4.3.5.2 G54-59.3 User Coordinates

Normally you use the G54 Coordinate System. When an offset is applied to a current user coordinate system a small blue ball with lines will be at the [machine origin](#) when your DRO is displaying *Position: Relative Actual* in Axis. If your offsets are temporary use the Zero Coordinate System from the Machine menu or program `G10 L2 P1 X0 Y0 Z0` at the end of your G-code file. Change the *P* number to suit the coordinate system you wish to clear the offset in.

- Offsets stored in a user coordinate system are retained when LinuxCNC is shut down.
- Using the *Touch Off* button in Axis sets an offset for the chosen User Coordinate System.

4.3.5.3 When You Are Lost

If you're having trouble getting 0,0,0 on the DRO when you think you should, you may have some offsets programmed in and need to remove them.

- Move to the Machine origin with `G53 G0 X0 Y0 Z0`
- Clear any G92 offset with `G92.1`
- Use the G54 coordinate system with `G54`
- Set the G54 coordinate system to be the same as the machine coordinate system with `G10 L2 P1 X0 Y0 Z0 R0`
- Turn off tool offsets with `G49`
- Turn on the Relative Coordinate Display from the menu

Now you should be at the machine origin X0 Y0 Z0 and the relative coordinate system should be the same as the machine coordinate system.

4.3.6 Machine Configurations

The following diagram shows a typical mill showing direction of travel of the tool and the mill table and limit switches. Notice how the mill table moves in the opposite direction of the Cartesian coordinate system arrows shown by the *Tool Direction* image. This makes the *tool* move in the correct direction in relation to the material.

Note also the position of the limit switches and the direction of activation of their cams. Several combinations are possible, for example it is possible (contrary to the drawing) to place a single fixed limit switch in the middle of the table and two mobile cams to activate it. In this case the limits will be reversed, +X will be on the right of the table and -X on the left. This inversion does not change anything from the point of view of the direction of movement of the tool.

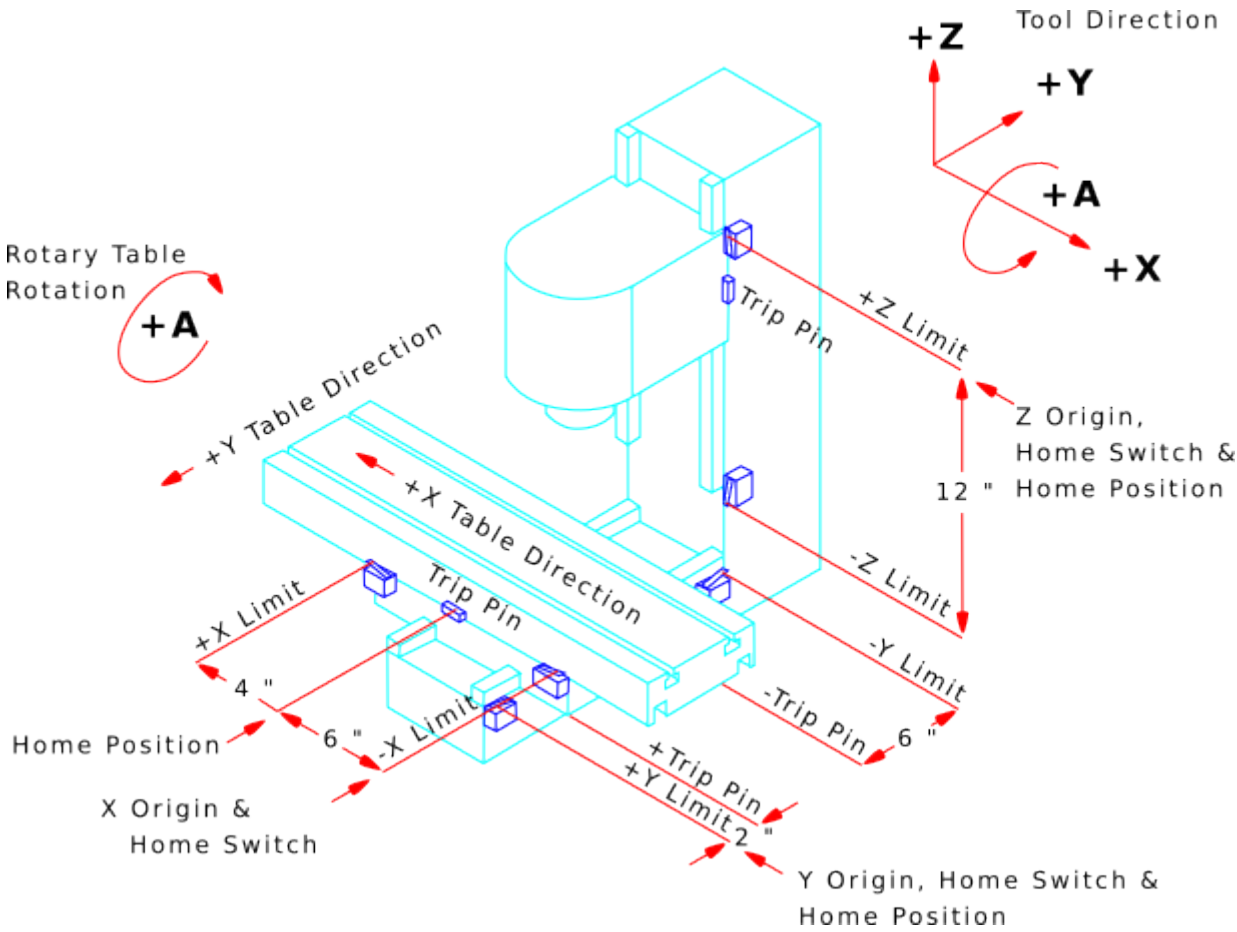


Figure 4.11: Typical Mill Configuration

The following diagram shows a typical lathe showing direction of travel of the tool and limit switches.

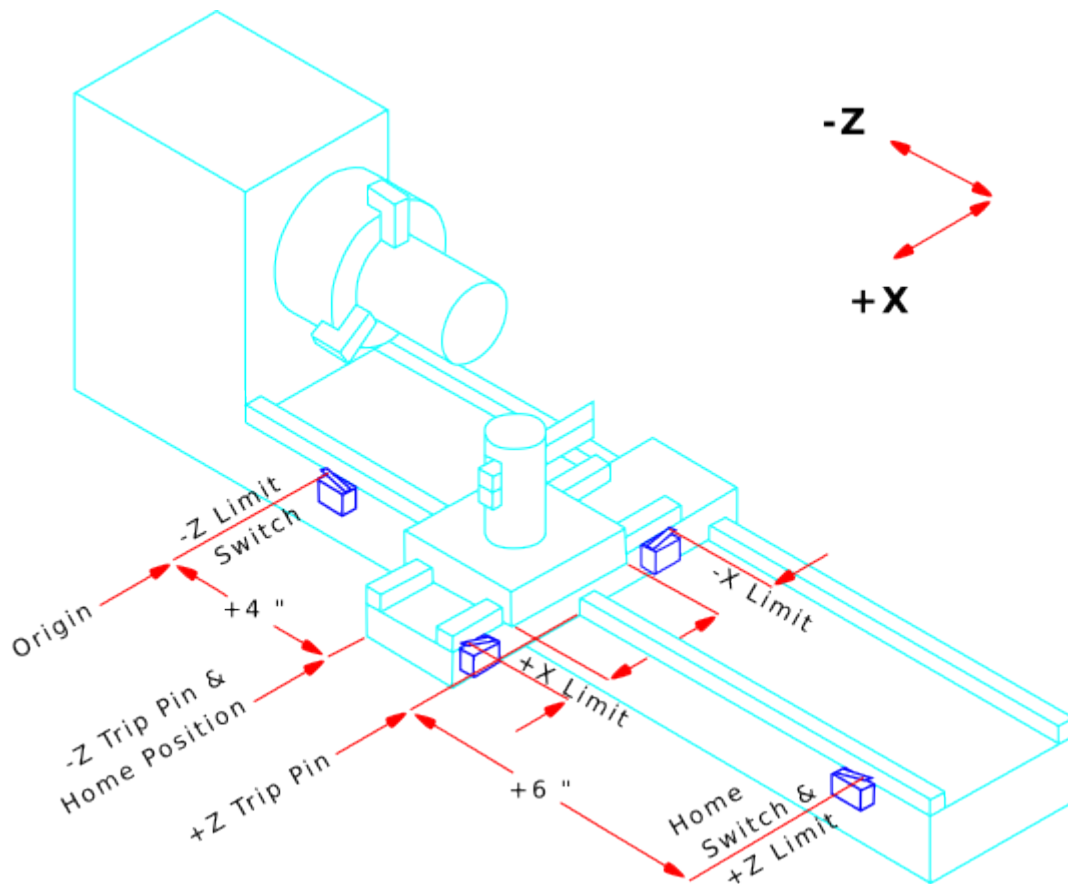


Figure 4.12: Typical Lathe Configuration

4.4 Starting LinuxCNC

4.4.1 Running LinuxCNC

LinuxCNC is started with the script file *linuxcnc*.

```
linuxcnc [options] [<ini-file>]
```

linuxcnc script options

linuxcnc: Run LINUXCNC

Usage:

```
$ linuxcnc -h
This help
```

```
$ linuxcnc [Options]
Choose the configuration inifile graphically
```

```
$ linuxcnc [Options] path/to/your_ini_file
Name the configuration inifile using its path
```

```
$ linuxcnc [Options] -l
Use the previously used configuration inifile
```

Options:

- d: Turn on "debug" mode
- v: Turn on "verbose" mode
- k: Continue in the presence of errors in .hal files
- t "tpmodulename [parameters]"
 - specify custom trajectory_planning_module
 - overrides optional ini setting [TRAJ]TPMOD
- m "homemodulename [parameters]"
 - specify custom homing_module
 - overrides optional ini setting [EMCMOT]HOMEMOD
- H "dirname": search dirname for Halfiles before searching
 - ini directory and system library:
 - /home/git/linuxcnc-dev/lib/hallib

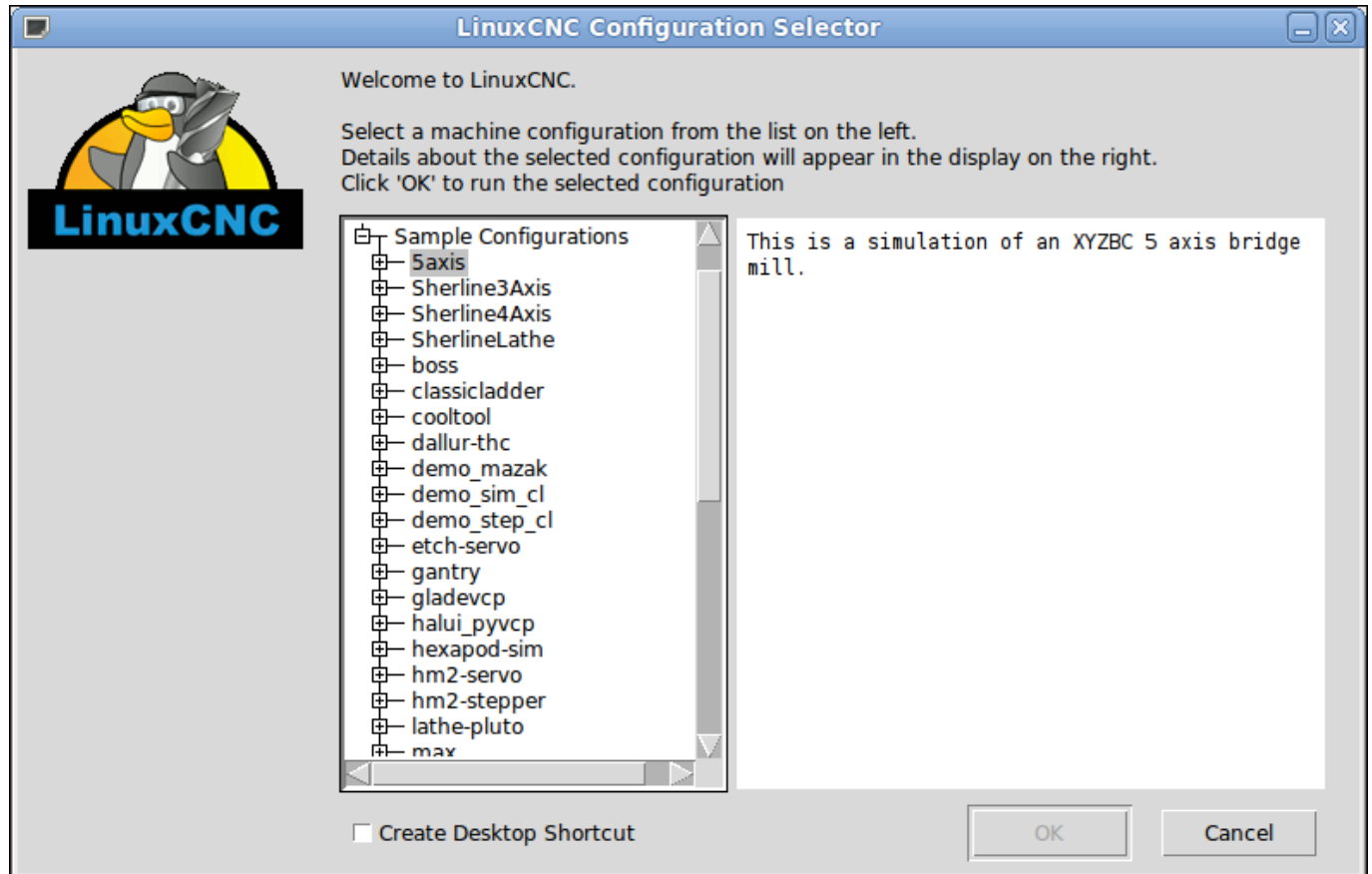
Note:

The -H "dirname" option may be specified multiple times

If the linuxcnc script is passed an ini file it reads the ini file and starts LinuxCNC. The ini file [HAL] section specifies the order of loading up HAL files if more than one is used. Once the HAL=xxx.hal files are loaded then the GUI is loaded then the POSTGUI=.xxx.hal file is loaded. If you create PyVCP or GladeVCP objects with HAL pins you must use the postgui HAL file to make any connections to those pins. See the [\[HAL\]](#) section of the INI configuration for more information.

4.4.1.1 Configuration Selector

If no ini file is passed to the linuxcnc script it loads the configuration selector so you can choose and save a sample configuration. Once a sample configuration has been saved it can be modified to suit your application. The configuration files are saved in linuxcnc/configs directory.



4.5 CNC Machine Overview

This section gives a brief description of how a CNC machine is viewed from the input and output ends of the Interpreter.

4.5.1 Mechanical Components

A CNC machine has many mechanical components that may be controlled or may affect the way in which control is exercised. This section describes the subset of those components that interact with the Interpreter. Mechanical components that do not interact directly with the Interpreter, such as the jog buttons, are not described here, even if they affect control.

4.5.1.1 Axes

Any CNC machine has one or more Axes. Different types of CNC machines have different combinations. For instance, a *4-axis milling machine* may have XYZA or XYZB axes. A lathe typically has XZ axes. A foam-cutting machine may have XYUV axes. In LinuxCNC, the case of a *XYZ gantry machine* with two motors for one axis is better handled by kinematics rather than by a second linear axis.

Note

If the motion of mechanical components is not independent, as with hexapod machines, the RS274/NGC language and the canonical machining functions will still be usable, as long as the lower levels of control know how to control the actual mechanisms to produce the same relative motion of tool and workpiece as would be produced by independent axes. This is called *kinematics*.

Note

With LinuxCNC, the case of the *XYZ gantry machine* with two motors for one axis is better handled by the kinematics than by an additional linear axis.

Primary Linear Axes The X, Y, and Z axes produce linear motion in three mutually orthogonal directions.

Secondary Linear Axes The U, V, and W axes produce linear motion in three mutually orthogonal directions. Typically, X and U are parallel, Y and V are parallel, and Z and W are parallel.

Rotational Axes The A, B and C axes produce angular motion (rotation). Typically, A rotates around a line parallel to X, B rotates around a line parallel to Y, and C rotates around a line parallel to Z.

4.5.1.2 Spindle

A CNC machine typically has a spindle which holds one cutting tool, probe, or the material in the case of a lathe. The spindle may or may not be controlled by the CNC software. LinuxCNC offers support for up to 8 spindles, which can be individually controlled and can run simultaneously at different speeds and in different directions.

4.5.1.3 Coolant

Flood coolant and mist coolant may each be turned on independently. The RS274/NGC language turns them off together see section [M7 M8 M9](#).

4.5.1.4 Feed and Speed Override

A CNC machine can have separate feed and speed override controls, which let the operator specify that the actual feed rate or spindle speed used in machining at some percentage of the programmed rate.

4.5.1.5 Block Delete Switch

A CNC machine can have a block delete switch. See the [Block Delete](#) section.

4.5.1.6 Optional Program Stop Switch

A CNC machine can have an optional program stop switch. See the [Optional Program Stop](#) section.

4.5.2 Control and Data Components

4.5.2.1 Linear Axes

The X, Y, and Z axes form a standard right-handed coordinate system of orthogonal linear axes. Positions of the three linear motion mechanisms are expressed using coordinates on these axes.

The U, V and W axes also form a standard right-handed coordinate system. X and U are parallel, Y and V are parallel, and Z and W are parallel (when A, B, and C are rotated to zero).

4.5.2.2 Rotational Axes

The rotational axes are measured in degrees as wrapped linear axes in which the direction of positive rotation is counterclockwise when viewed from the positive end of the corresponding X, Y, or Z-axis. By *wrapped linear axis*, we mean one on which the angular position increases without limit (goes towards plus infinity) as the axis turns counterclockwise and decreases without limit (goes towards minus infinity) as the axis turns clockwise. Wrapped linear axes are used regardless of whether or not there is a mechanical limit on rotation.

Clockwise or counterclockwise is from the point of view of the workpiece. If the workpiece is fastened to a turntable which turns on a rotational axis, a counterclockwise turn from the point of view of the workpiece is accomplished by turning the turntable in a direction that (for most common machine configurations) looks clockwise from the point of view of someone standing next to the machine. ³

4.5.2.3 Controlled Point

The controlled point is the point whose position and rate of motion are controlled. When the tool length offset is zero (the default value), this is a point on the spindle axis (often called the gauge point) that is some fixed distance beyond the end of the spindle, usually near the end of a tool holder that fits into the spindle. The location of the controlled point can be moved out along the spindle axis by specifying some positive amount for the tool length offset. This amount is normally the length of the cutting tool in use, so that the controlled point is at the end of the cutting tool. On a lathe, tool length offsets can be specified for X and Z axes, and the controlled point is either at the tool tip or slightly outside it (where the perpendicular, axis-aligned lines touched by the *front* and *side* of the tool intersect).

³If the parallelism requirement is violated, the system builder will have to say how to distinguish clockwise from counterclockwise.

4.5.2.4 Coordinated Linear Motion

To drive a tool along a specified path, a machining center must often coordinate the motion of several axes. We use the term *coordinated linear motion* to describe the situation in which, nominally, each axis moves at constant speed and all axes move from their starting positions to their end positions at the same time. If only the X, Y, and Z axes (or any one or two of them) move, this produces motion in a straight line, hence the word *linear* in the term. In actual motions, it is often not possible to maintain constant speed because acceleration or deceleration is required at the beginning and/or end of the motion. It is feasible, however, to control the axes so that, at all times, each axis has completed the same fraction of its required motion as the other axes. This moves the tool along same path, and we also call this kind of motion coordinated linear motion.

Coordinated linear motion can be performed either at the prevailing feed rate, or at traverse rate, or it may be synchronized to the spindle rotation. If physical limits on axis speed make the desired rate unobtainable, all axes are slowed to maintain the desired path.

4.5.2.5 Feed Rate

The rate at which the controlled point moves is nominally a steady rate which may be set by the user. In the Interpreter, the feed rate is interpreted as follows (unless *inverse time feed* or *feed per revolution* modes are being used, in which case see section [G93-G94-G95-Mode](#)).

1. If any of XYZ are moving, F is in units per minute in the XYZ cartesian system, and all other axes (ABCUVW) move so as to start and stop in coordinated fashion.
2. Otherwise, if any of UVW are moving, F is in units per minute in the UVW cartesian system, and all other axes (ABC) move so as to start and stop in coordinated fashion.
3. Otherwise, the move is pure rotary motion and the F word is in rotary units in the ABC *pseudo-cartesian* system.

4.5.2.6 Cooling

Flood or droplets cooling can be enabled separately. RS274/NGC language stops them together. See section about [cooling control](#).

4.5.2.7 Dwell

A machining center may be commanded to dwell (i.e., keep all axes unmoving) for a specific amount of time. The most common use of dwell is to break and clear chips, so the spindle is usually turning during a dwell. Regardless of the Path Control Mode (see section [Path Control](#)) the machine will stop exactly at the end of the previous programmed move, as though it was in exact path mode.

4.5.2.8 Units

Units used for distances along the X, Y, and Z axes may be measured in millimeters or inches. Units for all other quantities involved in machine control cannot be changed. Different quantities use different specific units. Spindle speed is measured in revolutions per minute. The positions of rotational axes are measured in degrees. Feed rates are expressed in current length units per minute, or degrees per minute, or length units per spindle revolution, as described in section [G93 G94 G95](#).

4.5.2.9 Current Position

The controlled point is always at some location called the *current position*, and the controller always knows where that is. The numbers representing the current position must be adjusted in the absence of any axis motion if any of several events take place:

1. Length units are changed.
2. Tool length offset is changed.
3. Coordinate system offsets are changed.

4.5.2.10 Selected Plane

There is always a *selected plane*, which must be the XY-plane, the YZ-plane, or the XZ-plane of the machining center. The Z-axis is, of course, perpendicular to the XY-plane, the X-axis to the YZ-plane, and the Y-axis to the XZ-plane.

4.5.2.11 Tool Carousel

Zero or one tool is assigned to each slot in the tool carousel.

4.5.2.12 Tool Change

A machining center may be commanded to change tools.

4.5.2.13 Pallet Shuttle

The two pallets may be exchanged by command.

4.5.2.14 Speed Override

The speed override buttons can be activated (they function normally) or rendered inoperative (they no longer have any effect). The RS274/NGC language has a command that activates all the buttons and another that disables them. See inhibition and activation [speed correctors](#). See also [here for further details](#).

4.5.2.15 Path Control Mode

The machining center may be put into any one of three path control modes:

exact stop mode

In exact stop mode, the machine stops briefly at the end of each programmed move.

exact path mode

In exact path mode, the machine follows the programmed path as exactly as possible, slowing or stopping if necessary at sharp corners of the path.

continuous mode

In continuous mode, sharp corners of the path may be rounded slightly so that the feed rate may be kept up (but by no more than the tolerance, if specified).

See sections [G61](#) and [G64](#).

4.5.3 Interpreter Interaction with Switches

The Interpreter interacts with several switches. This section describes the interactions in more detail. In no case does the Interpreter know what the setting of any of these switches is.

4.5.3.1 Feed and Speed Override Switches

The Interpreter will interpret RS274/NGC commands which enable *M48* or disable *M49* the feed and speed override switches. For certain moves, such as the traverse out of the end of a thread during a threading cycle, the switches are disabled automatically.

LinuxCNC reacts to the speed and feed override settings when these switches are enabled.

See the [M48 M49 Override](#) section for more information.

4.5.3.2 Block Delete Switch

If the block delete switch is on, lines of G-code which start with a slash (the block delete character) are not interpreted. If the switch is off, such lines are interpreted. Normally the block delete switch should be set before starting the NGC program.

4.5.3.3 Optional Program Stop Switch

If this switch is on and an M1 code is encountered, program execution is paused.

4.5.4 Tool Table

A tool table is required to use the Interpreter. The file tells which tools are in which tool changer slots and what the size and type of each tool is. The name of the tool table is defined in the INI file:

```
[EMCIO]

# tool table file
TOOL_TABLE = tooltable.tbl
```

The default filename probably looks something like the above, but you may prefer to give your machine its own tool table, using the same name as your INI file, but with a tbl extension:

```
TOOL_TABLE = acme_300.tbl
```

or:

```
TOOL_TABLE = EMC-AXIS-SIM.tbl
```

For more information on the specifics of the tool table format, see the [Tool Table Format](#) section.

4.5.5 Parameters

In the RS274/NGC language view, a machining center maintains an array of numerical parameters defined by a system definition (RS274NGC_MAX_PARAMETERS). Many of them have specific uses especially in defining coordinate systems. The number of numerical parameters can increase as development adds support for new parameters. The parameter array persists over time, even if the machining center is powered down. LinuxCNC uses a parameter file to ensure persistence and gives

the Interpreter the responsibility for maintaining the file. The Interpreter reads the file when it starts up, and writes the file when it exits.

All parameters are available for use in G-code programs.

The format of a parameter file is shown in the following table. The file consists of any number of header lines, followed by one blank line, followed by any number of lines of data. The Interpreter skips over the header lines. It is important that there be exactly one blank line (with no spaces or tabs, even) before the data. The header line shown in the following table describes the data columns, so it is suggested (but not required) that that line always be included in the header.

The Interpreter reads only the first two columns of the table. The third column, *Comment*, is not read by the Interpreter.

Each line of the file contains the index number of a parameter in the first column and the value to which that parameter should be set in the second column. The value is represented as a double-precision floating point number inside the Interpreter, but a decimal point is not required in the file. All of the parameters shown in the following table are required parameters and must be included in any parameter file, except that any parameter representing a rotational axis value for an unused axis may be omitted. An error will be signaled if any required parameter is missing. A parameter file may include any other parameter, as long as its number is in the range 1 to 5400. The parameter numbers must be arranged in ascending order. An error will be signaled if not. Any parameter included in the file read by the Interpreter will be included in the file it writes as it exits. The original file is saved as a backup file when the new file is written. Comments are not preserved when the file is written.

Table 4.1: Parameter File Format

Parameter Number	Parameter Value	Comment
5161	0.0	G28 Home X
5162	0.0	G28 Home Y

See the [Parameters](#) section for more information.

4.6 Running LinuxCNC

4.6.1 Invoking LinuxCNC

After installation, LinuxCNC starts just like any other Linux program: run it from the [terminal](#) by issuing the command `linuxcnc`, or select it in the *Applications -> CNC* menu.

4.6.2 Configuration Launcher

When starting LinuxCNC (from the CNC menu or from the command line without specifying an INI file) the Configuration Selector dialog starts.

The Configuration Selector dialog allows the user to pick one of their existing configurations (My Configurations) or select a new one (from the Sample Configurations) to be copied to their home directory. Copied configurations will appear under My Configurations on the next invocation of the Configuration Selector.

The Configuration Selector offers a selection of configurations organized:

- *My Configurations* - User configurations located in `linuxcnc/configs` in your home directory.

- *Sample Configurations* - Sample configurations, when selected, are copied to linuxcnc/configs. Once a sample configuration was copied to your local directory, the launcher will offer it as *My Configurations*. The names under which these local configurations are presented correspond to the names of the directories within the configs/ directory:
 - *sim* - Configurations that include simulated hardware. These can be used for testing or learning how LinuxCNC works.
 - *by_interface* - Configurations organized by GUI.
 - *by_machine* - Configurations organized by machine.
 - *apps* - Applications that do not require starting linuxcnc but may be useful for testing or trying applications like [PyVCP](#) or [GladeVCP](#).
 - *attic* - Obsolete or historical configurations.

The sim configurations are often the most useful starting point for new users and are organized around supported GUIs:

- *axis* - Keyboard and Mouse GUI
- *gmoccapy* - Touch Screen GUI
- *gscreen* - Touch Screen GUI
- *low_graphics* - Keyboard GUI
- *pyvcp_demo* - Paneles de Control Virtuales Python
- *qtvcp_screens* - Guis diseñadas con Qt5 y Python
- *tklinuxcnc* - Keyboard and Mouse Gui(no longer maintained)
- *touchy* - Touch Screen GUI

A GUI configuration directory may contain subdirectories with configurations that illustrate special situations or the embedding of other applications.

The *by_interface* configurations are organized around common, supported interfaces like:

- general mechatronics
- mesa
- parport
- pico
- pluto
- servotogo
- vigilant
- vitalsystems

Related hardware may be required to use these configurations as starting points for a system.

The *by_machine* configurations are organized around complete, known systems like:

- boss
 - cooltool
 - plasmac
-

- scortbot erIII
- sherline
- smithy
- tormach

A complete system may be required to use these configurations.

The *apps items* are typically 1) utilities that don't require starting linuxcnc or 2) demonstrations of applications that can be used with linuxcnc:

- info - creates a file with system information that may be useful for problem diagnosis.
- gladevcp - Example GladeVCP applications.
- halrun - Starts halrun in an [terminal](#).
- latency - Applications to investigate latency
 - latency-test - standard test
 - latency-plot - stripchart
 - latency-histogram - histogram
- parport - Applications to test parport.
- pyvcp - Example pyvcp applications.
- xhc-hb04 - Applications to test an xhc-hb04 USB wireless MPG

Note

Under the Apps directory, only applications that are usefully modified by the user are offered for copying to the user's directory.



Figure 4.13: LinuxCNC Configuration Selector

Click any of the listed configurations to display specific information about it. Double-click a configuration or click OK to start the configuration.

Select *Create Desktop Shortcut* and then click *OK* to add an icon on the Ubuntu desktop to directly launch this configuration without showing the Configuration Selector screen.

When you select a configuration from the Sample Configurations section, it will automatically place a copy of that config in the `~/linuxcnc/configs` directory.

4.6.3 Next steps in configuration

After finding the sample configuration that uses the same interface hardware as your machine (or a simulator configuration), and saving a copy to your home directory, you can customize it according to the details of your machine. Refer to the Integrator Manual for topics on configuration.

4.6.4 Simulator Configurations

All configurations listed under Sample Configurations/sim are intended to run on any computer. No specific hardware is required and real-time support is not needed.

These configurations are useful for studying individual capabilities or options. The sim configurations are organized according to the graphical user interface used in the demonstration. The directory for axis contains the most choices and subdirectories because it is the most tested GUI. The capabilities demonstrated with any specific GUI may be available in other GUIs as well.

4.6.5 Configuration Resources

The Configuration Selector copies all files needed for a configuration to a new subdirectory of `~/linuxcnc/configs` (equivalently: `/home/username/linuxcnc/configs`). Each created directory will include at least one INI file (`inifilename.ini`) that is used to describe a specific configuration.

File resources within the copied directory will typically include one or more INI file (`filename.ini`) for related configurations and a tool table file (`toolfilename.tbl`). Additionally, resources may include HAL files (`filename.hal`, `filename.tcl`), a README file for describing the directory, and configuration specific information in a text file named after a specific configuration (`inifilename.txt`). That latter two files are displayed when using the Configuration Selector.

The supplied sample configurations may specify the parameter `HALFILE` (`filename.hal`) in the configuration INI file that are not present in the copied directory because they are found in the system HAL file library. These files can be copied to the user configuration directory and altered as required by the user for modification or test. Since the user configuration directory is searched first when finding HAL files, local modifications will then prevail.

The Configuration selector makes a symbolic link in the user configuration directory (named `hallib`) that points to the system HAL file library. This link simplifies copying a library file. For example, to copy the library `core_sim.hal` file in order to make local modifications:

```
cd ~/linuxcnc/configs/name_of_configuration
cp hallib/core_sim.hal core_sim.hal
```

4.7 Stepper Configuration Wizard

4.7.1 Introduction

LinuxCNC is capable of controlling a wide range of machinery using many different hardware interfaces.

StepConf is a program that generates configuration files for LinuxCNC for a specific class of CNC machine: those that are controlled via a *standard parallel port*, and controlled by signals of type *step & direction*.

StepConf is installed when you install LinuxCNC and is in the CNC menu.

StepConf places a file in the `linuxcnc/config` directory to store the choices for each configuration you create. When you change something, you need to pick the file that matches your configuration name. The file extension is `.stepconf`.

The StepConf Wizard works best with at least 800 x 600 screen resolution.

4.7.2 Start Page

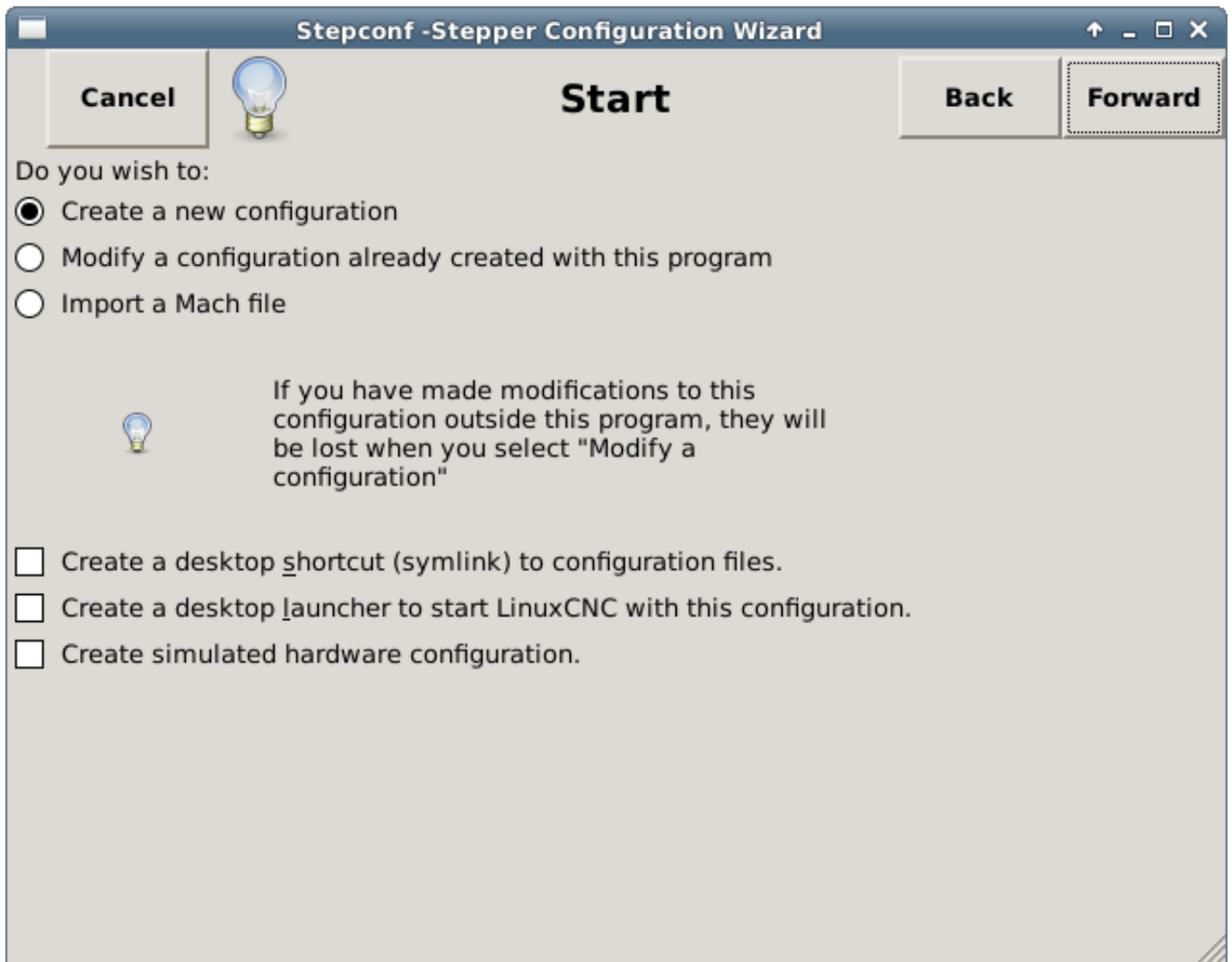


Figure 4.14: StepConf Entry Page

The three first radio buttons are self-explanatory:

- *Create New* - Creates a fresh configuration.
- *Modify* - Modify an existing configuration. After selecting this a file picker pops up so you can select the .stepconf file for modification. If you made any modifications to the main .hal or the .ini file these will be lost. Modifications to custom.hal and custom_postgui.hal will not be changed by the StepConf Wizard. StepConf will highlight the lastconf that was built.
- *Import* - Import a Mach configuration file and attempt to convert it to a linuxcnc config file. After the import, you will go through the pages of StepConf to confirm/modify the entries. The original mach xml file will not be changed.

These next options will be recorded in a preference file for the next run of StepConf.

- *Create Desktop Shortcut* - This will place a link on your desktop to the files.

- *Create Desktop Launcher* - This will place a launcher on your desktop to start your application.
- *Create Simulated Hardware* - This allows you to build a config for testing, even if you don't have the actual hardware.

4.7.3 Basic Information

Stepconf -Stepper Configuration Wizard

Base Information

Machine Name: my-mill

Configuration directory: ~/linuxcnc/configs/my-mill

Axis configuration: XYZ

Reset Default machine units: Inch

Driver characteristics: (Multiply by 1000 for times specified in μ s or microseconds)

Driver type: Other

▼ Driver Timing Settings

Step Time: 5000 ns

Step Space: 5000 ns

Direction Hold: 20000 ns

Direction Setup: 20000 ns

☒ One Parport ☐ Two Parports

Base Period Maximum Jitter: 15000 ns

Test Base Period Jitter

Min Base Period: 30000 ns

Max step rate: 33333 Hz

Figure 4.15: Basic Information Page

- *Create Simulated Hardware* - This allows you to build a config for testing, even if you don't have the actual hardware.
- *Machine Name* - Choose a name for your machine. Use only uppercase letters, lowercase letters, digits, - and _.
- *Axis Configuration* - Choose XYZ (Mill), XYZA (4-axis mill) or XZ (Lathe).
- *Machine Units* - Choose Inch or mm. All subsequent entries will be in the chosen units. Changing this also changes the default values in the Axes section. If you change this after selecting values in any of the axes sections, they will be over-written by the default values of the selected units.

- *Driver Type* - If you have one of the stepper drivers listed in the pull down box, choose it. Otherwise, select *Other* and find the timing values in your driver's data sheet and enter them as *nano seconds* in the *Driver Timing Settings*. If the data sheet gives a value in microseconds, multiply by 1000. For example, enter 4.5 μ s as 4500 ns.

A list of some popular drives, along with their timing values, is on the LinuxCNC.org Wiki under [Stepper Drive Timing](#).

Additional signal conditioning or isolation such as optocouplers and RC filters on break out boards can impose timing constraints of their own, in addition to those of the driver. You may find it necessary to add some time to the drive requirements to allow for this.

The LinuxCNC Configuration Selector has configs for Sherline already configured. * *Step Time* - How long the step pulse is *on* in nano seconds. If your not sure about this setting a value of 20,000 will work with most drives. * *Step Space* - Minimum time between step pulses in nano seconds. If your not sure about this setting a value of 20,000 will work with most drives. * *Direction Hold* - How long the direction pin is held after a change of direction in nanoseconds. If your not sure about this setting a value of 20,000 will work with most drives. * *Direction Setup* - How long before a direction change after the last step pulse in nanoseconds. If your not sure about this setting a value of 20,000 will work with most drives. * *One / Two Parport* - Select how many parallel port are to be configured. * *Base Period Maximum Jitter* - Enter the result of the Latency Test here. To run a latency test press the *Test Base Period Jitter* button. See the [Latency Test](#) section for more details. * *Max Step Rate* - StepConf automatically calculates the Max Step Rate based on the driver characteristics entered and the latency test result. * *Min Base Period* - StepConf automatically determines the Min Base Period based on the driver characteristics entered and latency test result.

4.7.4 Latency Test

While the test is running, you should *abuse* the computer. Move windows around on the screen. Surf the web. Copy some large files around on the disk. Play some music. Run an OpenGL program such as glxgears. The idea is to put the PC through its paces while the latency test checks to see what the worst case numbers are. Run the test at least a few minutes. The longer you run the test the better it will be at catching events that might occur at less frequent intervals. This is a test for your computer only, so no hardware needs to be connected to run the test.



Warning

Do not attempt run LinuxCNC while the latency test is running.

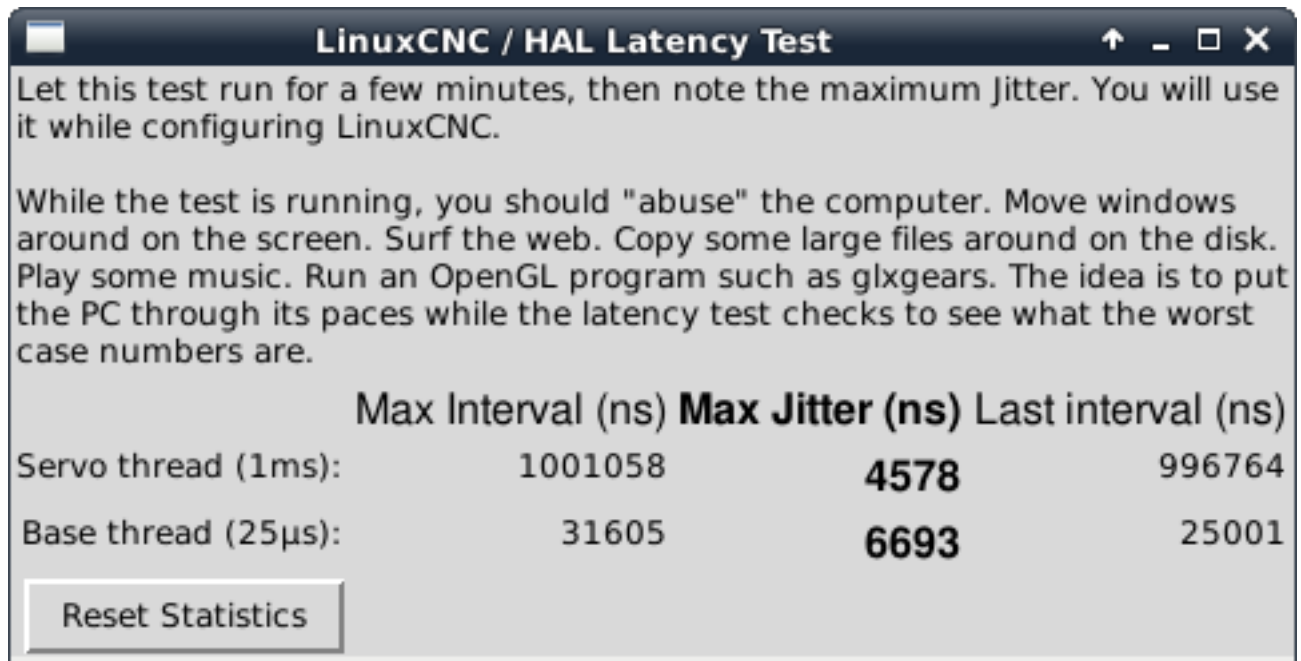


Figure 4.16: Latency Test

Latency is how long it takes the PC to stop what it is doing and respond to an external request. In our case, the request is the periodic *heartbeat* that serves as a timing reference for the step pulses. The lower the latency, the faster you can run the heartbeat, and the faster and smoother the step pulses will be.

Latency is far more important than CPU speed. The CPU isn't the only factor in determining latency. Motherboards, video cards, USB ports, SMI issues, and a number of other things can hurt the latency.

Troubleshooting SMI Issues (LinuxCNC.org Wiki)

Fixing Realtime problems caused by SMI on Ubuntu

<http://wiki.linuxcnc.org/cgi-bin/wiki.pl?FixingSMIIssues>

The important numbers are the *max jitter*. In the example above 9075 nanoseconds, or 9.075 microseconds, is the highest jitter. Record this number, and enter it in the Base Period Maximum Jitter box.

If your Max Jitter number is less than about 15-20 microseconds (15000-20000 nanoseconds), the computer should give very nice results with software stepping. If the max latency is more like 30-50 microseconds, you can still get good results, but your maximum step rate might be a little disappointing, especially if you use microstepping or have very fine pitch leadscrews. If the numbers are 100 μs or more (100,000 nanoseconds), then the PC is not a good candidate for software stepping. Numbers over 1 millisecond (1,000,000 nanoseconds) mean the PC is not a good candidate for LinuxCNC, regardless of whether you use software stepping or not.

4.7.5 Parallel Port Setup

Stepconf -Stepper Configuration Wizard

Parallel Port 1

Cancel Back Forward

Outputs (PC to Mill):		Invert	Inputs (Mill to PC):		Invert
Pin 1:	ESTOP Out	<input type="checkbox"/>	Pin 10:	Unused	<input type="checkbox"/>
Pin 2:	X Step	<input type="checkbox"/>	Pin 11:	Unused	<input type="checkbox"/>
Pin 3:	X Direction	<input type="checkbox"/>	Pin 12:	Unused	<input type="checkbox"/>
Pin 4:	Y Step	<input type="checkbox"/>	Pin 13:	Unused	<input type="checkbox"/>
Pin 5:	Y Direction	<input type="checkbox"/>	Pin 15:	Unused	<input type="checkbox"/>
Pin 6:	Z Step	<input type="checkbox"/>			
Pin 7:	Z Direction	<input type="checkbox"/>			
Pin 8:	A Step	<input type="checkbox"/>			
Pin 9:	A Direction	<input type="checkbox"/>			
Pin 14:	Spindle CW	<input type="checkbox"/>			
Pin 16:	Spindle PWM	<input type="checkbox"/>			
Pin 17:	Amplifier Enable	<input type="checkbox"/>			

Parport Base Address:

Output pinout presets:

Preset

Figure 4.17: Parallel Port Setup Page

You may specify the address as a hexadecimal (often 0x378) or as linux's default port number (probably 0)

For each pin, choose the signal which matches your parallel port pinout. Turn on the *invert* check box if the signal is inverted (0V for true/active, 5V for false/inactive).

- *Output pinout presets* - Automatically set pins 2 through 9 according to the Sherline standard (Direction on pins 2, 4, 6, 8) or the Xylotex standard (Direction on pins 3, 5, 7, 9).
- *Inputs and Outputs* - If the input or output is not used set the option to *Unused*.
- *External E Stop* - This can be selected from an input pin drop down box. A typical E Stop chain uses all normally closed contacts.
- *Homing & Limit Switches* - These can be selected from an input pin drop down box for most configurations.

- *Charge Pump* - If your driver board requires a charge pump signal select Charge Pump from the drop down list for the output pin you wish to connect to your charge pump input. The charge pump output is connected to the base thread by StepConf. The charge pump output will be about 1/2 of the maximum step rate shown on the Basic Machine Configuration page.
- *Plasma Arc Voltage* - If you require a Mesa THCAD to input a plasma arc voltage then select Plasma Arc Voltage from the list of output pins. This will enable a THCAD page during the setup procedure for the entry of the card parameters.

4.7.6 Parallel Port 2 Setup

Stepconf -Stepper Configuration Wizard

Parallel Port 2

Cancel Back Forward

Outputs (PC to Mill):		Invert	Inputs (Mill to PC):		Invert
Pin 1:	Unused	<input type="checkbox"/>	Pin 10:	Unused	<input type="checkbox"/>
Pin 2:	Unused	<input type="checkbox"/>	Pin 11:	Unused	<input type="checkbox"/>
Pin 3:	Unused	<input type="checkbox"/>	Pin 12:	Unused	<input type="checkbox"/>
Pin 4:	Unused	<input type="checkbox"/>	Pin 13:	Unused	<input type="checkbox"/>
Pin 5:	Unused	<input type="checkbox"/>	Pin 14:	Unused	<input type="checkbox"/>
Pin 6:	Unused	<input type="checkbox"/>	Pin 15:	Unused	<input type="checkbox"/>
Pin 7:	Unused	<input type="checkbox"/>			
Pin 8:	Unused	<input type="checkbox"/>			
Pin 9:	Unused	<input type="checkbox"/>			
Pin 16:	Unused	<input type="checkbox"/>			
Pin 17:	Unused	<input type="checkbox"/>			

1 Out


Figure 4.18: Parallel Port 2 Setup Page


The second Parallel port (if selected) can be configured and its pins assigned on this page. No step and direction signals can be selected. You may select in or out to maximize the number of input/output pins that are available. You may specify the address as a hexadecimal (often 0x378) or as linux's default port number (probably 1).

4.7.7 Axis Configuration

Stepconf - Stepper Configuration Wizard

Axis X

Cancel  Back Forward

Motor steps per revolution:  Test this axis

Driver Microstepping:

Pulley teeth (Motor:Leadscrew): :

Leadscrew Pitch: rev / in

Maximum Velocity: in / s

Maximum Acceleration: in / s²

Home location:

Table travel: to

Home Switch location:

Home Search velocity:

Home Latch direction:

Time to accelerate to max speed: 0.0333 s

Distance to accelerate to max speed: 0.0167 in

Pulse rate at max speed: 8000.0 Hz

Axis Scale: $200 \times 2 \times (1.0 \div 1.0) \times 20.000 =$ 8000.0 Steps / in

Figure 4.19: Axis Configuration Screen

- *Motor Steps Per Revolution* - The number of full steps per motor revolution. If you know how many degrees per step the motor is (e.g., 1.8 degree), then divide 360 by the degrees per step to find the number of steps per motor revolution.
- *Driver Microstepping* - The amount of microstepping performed by the driver. Enter 2 for half-stepping.
- *Pulley Ratio* - If your machine has pulleys between the motor and leadscrew, enter the ratio here. If not, enter 1:1.
- *Leadscrew Pitch* - Enter the pitch of the leadscrew here. If you chose *Inch* units, enter the number of threads per inch. If you chose *mm* units, enter the number of millimeters per revolution (e.g., enter 2 for 2mm/rev). If the machine travels in the wrong direction, enter a negative number here instead of a positive number, or invert the direction pin for the axis.
- *Maximum Velocity* - Enter the maximum velocity for the axis in units per second.

- *Maximum Acceleration* - The correct values for these items can only be determined through experimentation. See [Finding Maximum Velocity](#) to set the speed and [Finding Maximum Acceleration](#) to set the acceleration.
 - *Home Location* - The position the machine moves to after completing the homing procedure for this axis. For machines without home switches, this is the location the operator manually moves the machine to before pressing the Home button. If you combine the home and limit switches you must move off of the switch to the home position or you will get a joint limit error.
 - *Table Travel* - The range of travel for that axis based on the machine origin. The home location must be inside the *Table Travel* and not equal to one of the *Table Travel* values.
 - *Home Switch Location* - The location at which the home switch trips or releases relative to the machine origin. This item and the two below only appear when Home Switches were chosen in the Parallel Port Pinout. If you combine home and limit switches the home switch location can not be the same as the home position or you will get a joint limit error.
 - *Home Search Velocity* - The velocity to use when searching for the home switch. If the switch is near the end of travel, this velocity must be chosen so that the axis can decelerate to a stop before hitting the end of travel. If the switch is only closed for a short range of travel (instead of being closed from its trip point to one end of travel), this velocity must be chosen so that the axis can decelerate to a stop before the switch opens again, and homing must always be started from the same side of the switch. If the machine moves the wrong direction at the beginning of the homing procedure, negate the value of *Home Search Velocity*.
 - *Home Latch Direction* - Choose *Same* to have the axis back off the switch, then approach it again at a very low speed. The second time the switch closes, the home position is set. Choose *Opposite* to have the axis back off the switch and when the switch opens, the home position is set.
 - *Time to accelerate to max speed* - Time to reach maximum speed calculated from *Max Acceleration* and *Max Velocity*.
 - *Distance to accelerate to max speed* - Distance to reach maximum speed from a standstill.
 - *Pulse rate at max speed* - Information computed based on the values entered above. The greatest *Pulse rate at max speed* determines the *BASE_PERIOD*. Values above 20000Hz may lead to slow response time or even lockups (the fastest usable pulse rate varies from computer to computer)
 - *Axis SCALE* - The number that will be used in the INI file [SCALE] setting. This is how many steps per user unit.
 - *Test this axis* - This will open a window to allow testing for each axis. This can be used after filling out all the information for this axis.
-

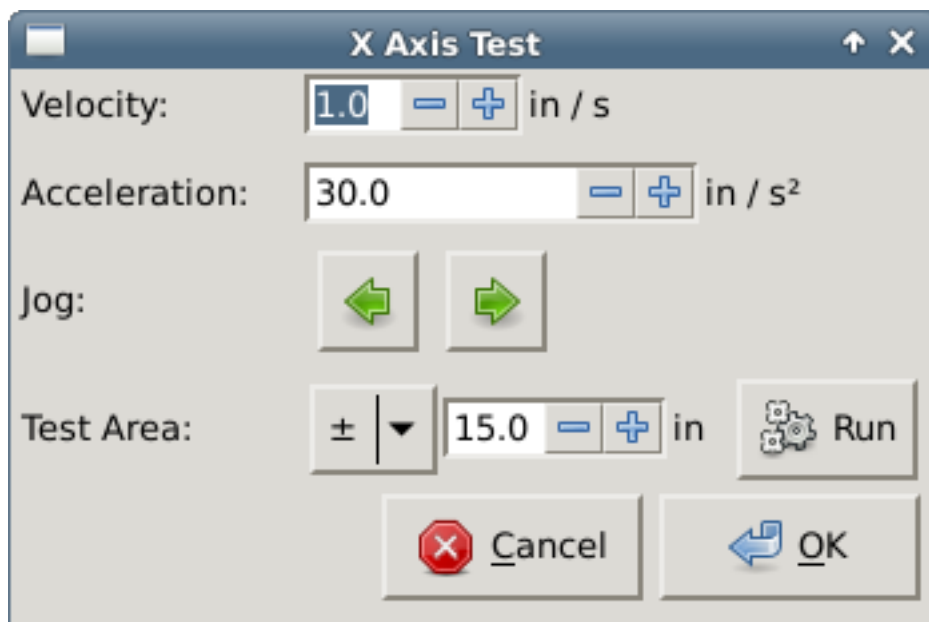


Figure 4.20: Axis Test

Test this axis is a basic tester that only outputs step and direction signals to try different values for acceleration and velocity.



Important

In order to use test this axis you have to manually enable the axis if this is required. If your driver has a charge pump you will have to bypass it. Test this axis does not react to limit switch inputs. Use with caution.

4.7.7.1 Finding Maximum Velocity

Begin with a low Acceleration (for example, **2 inches/s²** or **50 mm/s²**) and the velocity you hope to attain. Using the buttons provided, jog the axis to near the center of travel. Take care because with a low acceleration value, it can take a surprising distance for the axis to decelerate to a stop.

After gauging the amount of travel available, enter a safe distance in Test Area, keeping in mind that after a stall the motor may next start to move in an unexpected direction. Then click Run. The machine will begin to move back and forth along this axis. In this test, it is important that the combination of Acceleration and Test Area allow the machine to reach the selected Velocity and *cruise* for at least a short distance — the more distance, the better this test is. The formula $d = 0.5 * v * v/a$ gives the minimum distance required to reach the specified velocity with the given acceleration. If it is convenient and safe to do so, push the table against the direction of motion to simulate cutting forces. If the machine stalls, reduce the speed and start the test again.

If the machine did not obviously stall, click the *Run* button off. The axis now returns to the position where it started. If the position is incorrect, then the axis stalled or lost steps during the test. Reduce Velocity and start the test again.

If the machine doesn't move, stalls, or loses steps, no matter how low you turn Velocity, verify the following:

- Correct step waveform timings

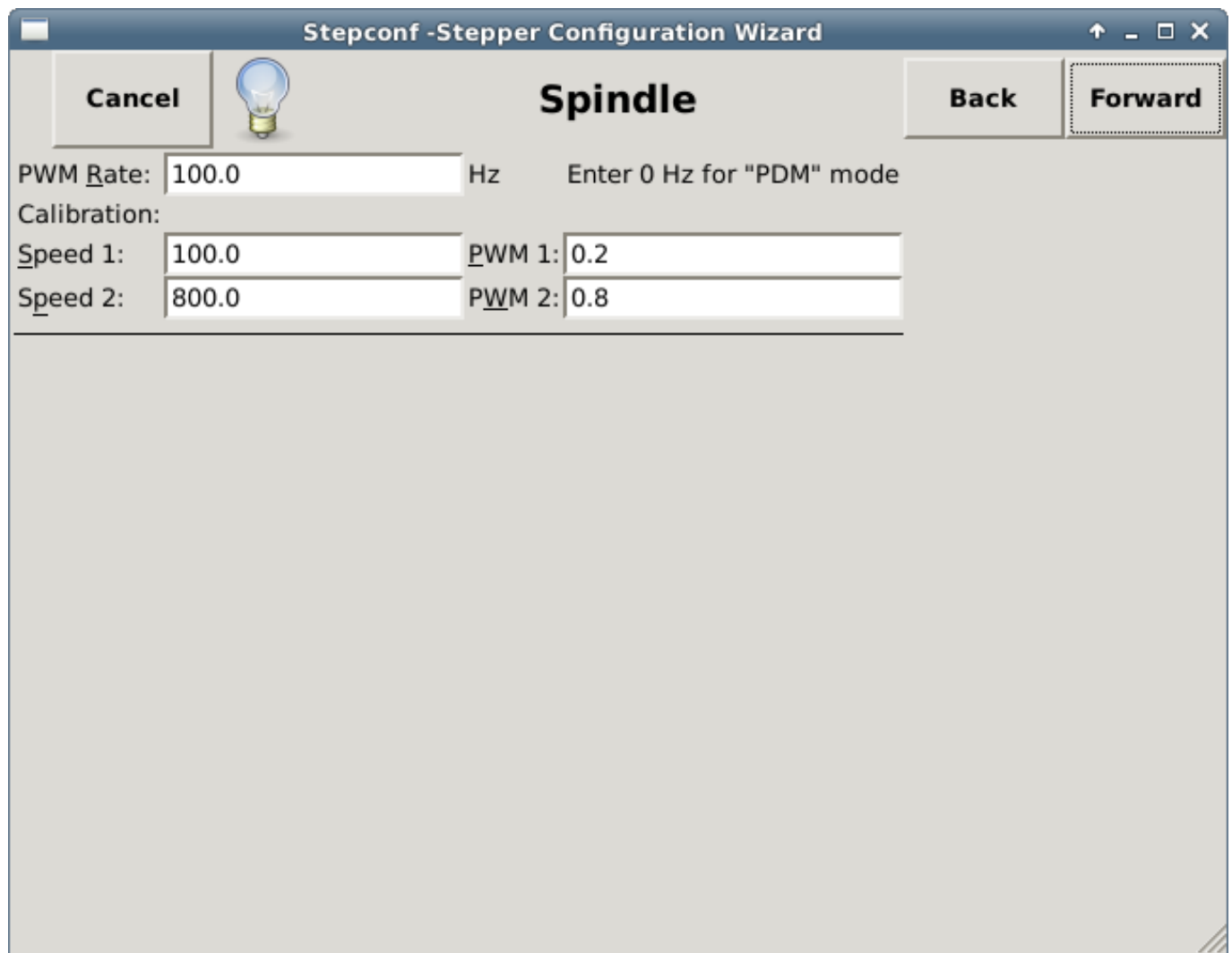
- Correct pinout, including *Invert* on step pins
- Correct, well-shielded cabling
- Physical problems with the motor, motor coupling, leadscrew, etc.

Once you have found a speed at which the axis does not stall or lose steps during this testing procedure, reduce it by 10% and use that as the axis *Maximum Velocity*.

4.7.7.2 Finding Maximum Acceleration

With the Maximum Velocity you found in the previous step, enter the acceleration value to test. Using the same procedure as above, adjust the Acceleration value up or down as necessary. In this test, it is important that the combination of Acceleration and Test Area allow the machine to reach the selected Velocity. Once you have found a value at which the axis does not stall or lose steps during this testing procedure, reduce it by 10% and use that as the axis Maximum Acceleration.

4.7.8 Spindle Configuration



The screenshot shows the 'Spindle' configuration page of the Stepconf -Stepper Configuration Wizard. The window has a title bar with standard Linux window controls. The page is divided into sections for configuration. At the top, there are 'Cancel', 'Back', and 'Forward' buttons. A lightbulb icon is positioned above the 'Spindle' title. The 'PWM Rate' is set to 100.0 Hz, with a note to 'Enter 0 Hz for "PDM" mode'. Below this, the 'Calibration' section contains two rows of settings: 'Speed 1' at 100.0 with 'PWM 1' at 0.2, and 'Speed 2' at 800.0 with 'PWM 2' at 0.8. The main area of the window is a large, empty text box for additional notes or comments.

Spindle Configuration	
PWM Rate:	100.0 Hz
Enter 0 Hz for "PDM" mode	
Calibration:	
Speed 1:	100.0
PWM 1:	0.2
Speed 2:	800.0
PWM 2:	0.8

Figure 4.21: Spindle Configuration Page

This page only appears when *Spindle PWM* is chosen in the *Parallel Port Pinout* page for one of the outputs.

4.7.8.1 Spindle Speed Control

If *Spindle PWM* appears on the pinout, the following information should be entered:

- *PWM Rate* - The *carrier frequency* of the PWM signal to the spindle. Enter 0 for PDM mode, which is useful for generating an analog control voltage. Refer to the documentation for your spindle controller for the appropriate value.
- *Speed 1 and 2, PWM 1 and 2* - The generated configuration file uses a simple linear relationship to determine the PWM value for a given RPM value. If the values are not known, they can be determined. For more information see [Determining Spindle Calibration](#).

4.7.8.2 Spindle-synchronized motion

When the appropriate signals from a spindle encoder are connected to LinuxCNC via HAL, LinuxCNC supports lathe threading. These signals are:

- *Spindle Index* - Is a pulse that occurs once per revolution of the spindle.
- *Spindle Phase A* - This is a pulse that occurs in multiple equally-spaced locations as the spindle turns.
- *Spindle Phase B (optional)* - This is a second pulse that occurs, but with an offset from Spindle Phase A. The advantages to using both A and B are direction sensing, increased noise immunity, and increased resolution.

If *Spindle Phase A* and *Spindle Index* appear on the pinout, the following information should be entered:

- *Use Spindle-At-Speed* - With encoder feedback one can choose to have linuxcnc wait for the spindle to reach the commanded speed before feed moves. Select this option and set the *close enough* scale.
- *Speed Display Filter Gain* - Setting for adjusting the stability of the visual spindle speed display.
- *Cycles per revolution* - The number of cycles of the *Spindle A* signal during one revolution of the spindle. This option is only enabled when an input has been set to *Spindle Phase A*
- *Maximum speed in thread* - The maximum spindle speed used in threading. For a high spindle RPM or a spindle encoder with high resolution, a low value of *BASE_PERIOD* is required.

4.7.8.3 Determining Spindle Calibration

Enter the following values in the Spindle Configuration page:

Speed 1:	0	PWM 1:	0
Speed 2:	1000	PWM 2:	1

Finish the remaining steps of the configuration process, then launch LinuxCNC with your configuration. Turn the machine on and select the MDI tab. Start the spindle turning by entering: *M3 S100*. Change the spindle speed by entering a different S-number: *S800*. Valid numbers (at this point) range from 1 to 1000.

For two different S-numbers, measure the actual spindle speed in RPM. Record the S-numbers and actual spindle speeds. Run StepConf again. For *Speed* enter the measured speed, and for *PWM* enter the S-number divided by 1000.

Because most spindle drivers are somewhat nonlinear in their response curves, it is best to:

- Make sure the two calibration speeds are not too close together in RPM.
- Make sure the two calibration speeds are in the range of speeds you will typically use while milling.

For instance, if your spindle will go from 0 RPM to 8000 RPM, but you generally use speeds from 400 RPM (10%) to 4000 RPM (100%), then find the PWM values that give 1600 RPM (40%) and 2800 RPM (70%).

4.7.9 Options

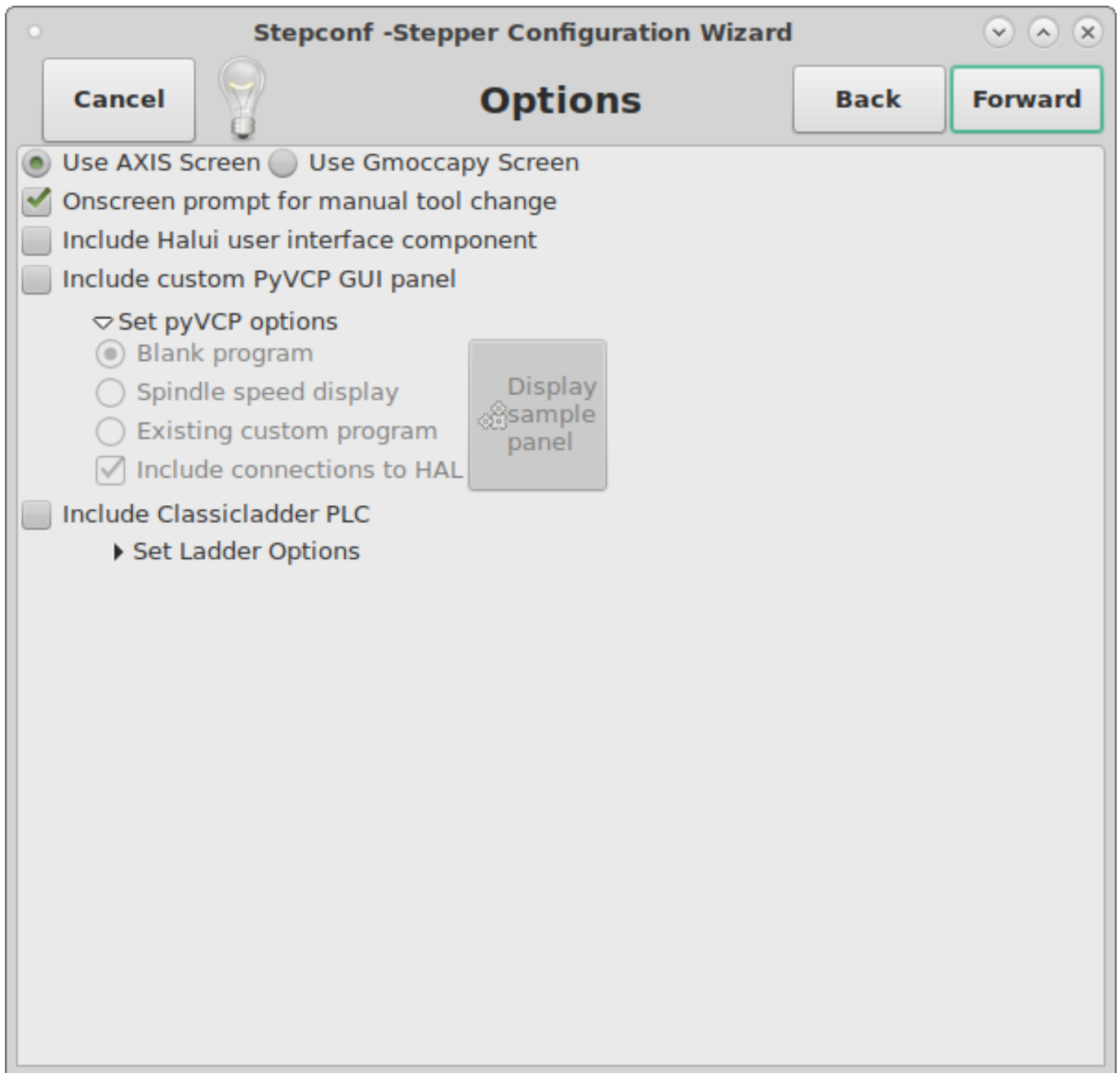


Figure 4.22: Advanced Options Configuration

- *Include Halui* - This will add the Halui user interface component. See the [HALUI Chapter](#) for more information on.
- *Include PyVCP* - This option adds the PyVCP panel base file or a sample file to work on. See the [PyVCP Chapter](#) for more information.
- *Include ClassicLadder PLC* - This option will add the ClassicLadder PLC (Programmable Logic Controller). See the [Classicladder Chapter](#) for more information.

The *home position* is the location within travel that the axis will be moved to at the end of the homing sequence. This value must be within the *soft limits*. In particular, the *home position* should never be exactly equal to a *soft limit*.

4.7.11.1 Operating without Limit Switches

A machine can be operated without limit switches. In this case, only the soft limits stop the machine from reaching the hard stop. Soft limits only operate after the machine has been homed.

4.7.11.2 Operating without Home Switches

A machine can be operated without home switches. If the machine has limit switches, but no home switches, it is best to use a limit switch as the home switch (e.g., choose *Minimum Limit + Home X* in the pinout). If the machine has no switches at all, or the limit switches cannot be used as home switches for another reason, then the machine must be homed *by eye* or by using match marks. Homing by eye is not as repeatable as homing to switches, but it still allows the soft limits to be useful.

4.7.11.3 Home and Limit Switch wiring options

The ideal wiring for external switches would be one input per switch. However, the PC parallel port only offers a total of 5 inputs, while there are as many as 9 switches on a 3-axis machine. Instead, multiple switches are wired together in various ways so that a smaller number of inputs are required.

The figures below show the general idea of wiring multiple switches to a single input pin. In each case, when one switch is actuated, the value seen on INPUT goes from logic HIGH to LOW. However, LinuxCNC expects a TRUE value when a switch is closed, so the corresponding *Invert* box must be checked on the pinout configuration page. The pull up resistor shown in the diagrams pulls the input high until the connection to ground is made and then the input goes low. Otherwise the input might float between on and off when the circuit is open. Typically for a parallel port you might use 47k.

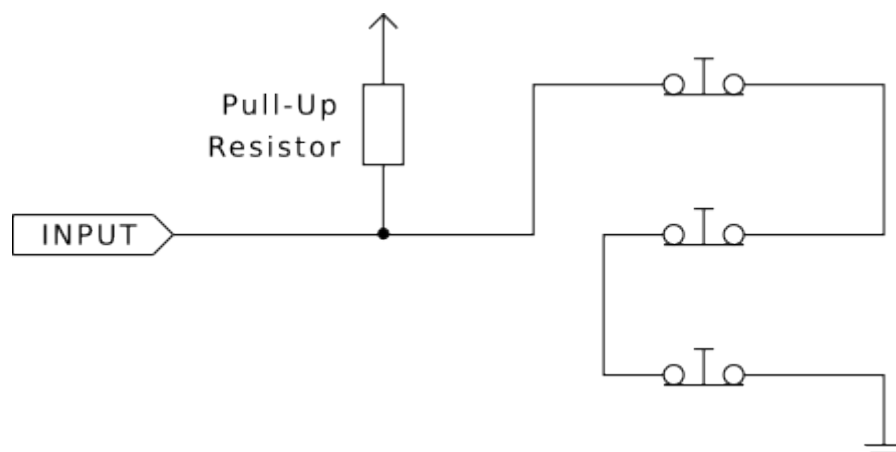


Figure 4.24: Normally Closed Switches (N/C) wiring in series (simplified diagram)

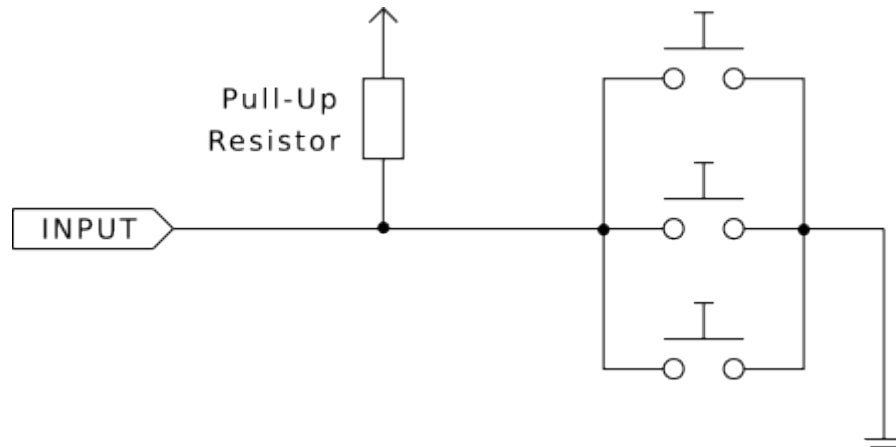


Figure 4.25: Normally Open Switches (N/O) wiring in parallel (simplified diagram)

The following combinations of switches are permitted in StepConf:

- Combine home switches for all axes
- Combine limit switches for all axes
- Combine both limit switches for one axis
- Combine both limit switches and the home switch for one axis
- Combine one limit switch and the home switch for one axis

The last two combinations are also appropriate when the type contact + home is used.

4.8 Mesa Configuration Wizard

PnCconf is made to help build configurations that utilize specific Mesa *Anything I/O* products.

It can configure closed loop servo systems or hardware stepper systems. It uses a similar *wizard* approach as Stepconf (used for software stepping, parallel port driven systems).

PnCconf is still in a development stage (Beta) so there are some bugs and lacking features. Please report bugs and suggestions to the LinuxCNC forum page or mail-list.

There are two trains of thought when using PnCconf:

One is to use PnCconf to always configure your system - if you decide to change options, reload PnCconf and allow it to configure the new options. This will work well if your machine is fairly standard and you can use custom files to add non standard features. PnCconf tries to work with you in this regard.

The other is to use PnCconf to build a config that is close to what you want and then hand edit everything to tailor it to your needs. This would be the choice if you need extensive modifications beyond PnCconf's scope or just want to tinker with / learn about LinuxCNC.

You navigate the wizard pages with the forward, back, and cancel buttons there is also a help button that gives some help information about the pages, diagrams and an output page.

Tip

PnCconf's help page should have the most up to date info and has additional details.

4.8.1 Step by Step Instructions

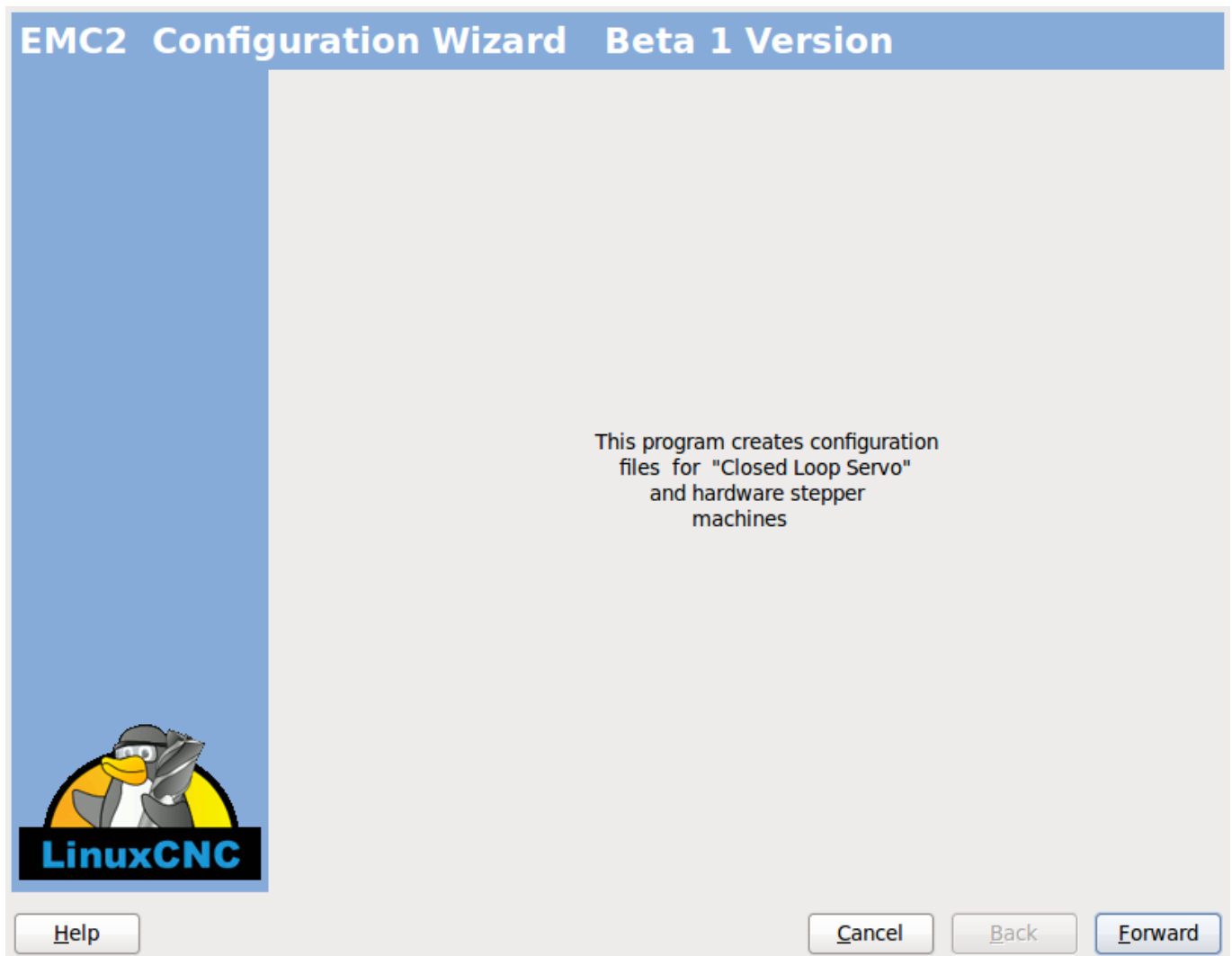


Figure 4.26: PnCConf Splash

4.8.2 Create or Edit

This allows you to select a previously saved configuration or create a new one. If you pick *Modify a configuration* and then press *Next* a file selection box will show. PnCConf preselects your last saved file. Choose the config you wish to edit. If you made any changes to the main HAL or INI files **PnCConf will over write** those files and those changes will be lost. Some files will not be over written and PnCConf places a note in those files. It also allows you to select desktop shortcut / launcher options. A desktop shortcut will place a folder icon on the desktop that points to your new configuration files. Otherwise you would have to look in your home folder under linuxcnc/configs.

A Desktop launcher will add an icon to the desktop for starting your config directly. You can also launch it from the main menu by using the Configuration Selector *LinuxCNC* found in CNC menu and selecting your config name.

4.8.3 Basic Machine Information

Basic machine information

Machine Basics

Machine Name: my EMC machine

Configuration directory: ~ / emc2 / configs / my EMC machine

Axis configuration: XYZ

Machine units: Inch

Computer Response Time

Actual Servo Period: 1000000 ns

Recommend servo period: 1000000

Test Base Period Jitter

I/O Control Ports/ Boards

☒ Mesa0 PCI / Parport Card: 5i20

☐ Mesa1 PCI / Parport Card: 5i20

☒ First Parport Address: 0x0278 Out

☐ Second Parport Address: Enter Address In

☐ Third Parport Address: Enter Address In

Add-on PCI Parport Address Search

GUI frontend list

☒ Axis

☐ TKemc

☐ Mini

☐ Touchy

Help Cancel Back Forward

Figure 4.27: PnCConf Basic

Machine Basics

If you use a name with spaces PnCconf will replace the spaces with underscore (as a loose rule Linux doesn't like spaces in names) Pick an axis configuration - this selects what type of machine you are building and what axes are available. The Machine units selector allows data entry of metric or imperial units in the following pages.

Tip

Defaults are not converted when using metric so make sure they are sane values!

Computer Response Time

The servo period sets the heart beat of the system. Latency refers to the amount of time the computer can be longer then that period. Just like a railroad, LinuxCNC requires everything

on a very tight and consistent time line or bad things happen. LinuxCNC requires and uses a *real time* operating system, which just means it has a low latency (lateness) response time when LinuxCNC requires its calculations and when doing LinuxCNC's calculations it cannot be interrupted by lower priority requests (such as user input to screen buttons or drawing etc).

Testing the latency is very important and a key thing to check early. Luckily by using the Mesa card to do the work that requires the fastest response time (encoder counting and PWM generation) we can endure a lot more latency than if we used the parallel port for these things. The standard test in LinuxCNC is checking the BASE period latency (even though we are not using a base period). If you press the *test base period jitter* button, this launches the latency test window (you can also load this directly from the applications/cnc panel). The test mentions to run it for a few minutes but the longer the better. Consider 15 minutes a bare minimum and overnight even better. At this time use the computer to load things, use the net, use USB etc we want to know the worst case latency and to find out if any particular activity hurts our latency. We need to look at base period jitter. Anything under 20000 is excellent - you could even do fast software stepping with the machine 20000 - 50000 is still good for software stepping and fine for us. 50000 - 100000 is really not that great but could still be used with hardware cards doing the fast response stuff. So anything under 100000 is usable to us. If the latency is disappointing or you get a bad hiccup periodically you may still be able to improve it.

Tip

There is a user compiled list of equipment and the latency obtained on the LinuxCNC wiki : <http://wiki.linuxcnc.org/cgi-bin/wiki.pl?Latency-Test> Please consider adding your info to the list. Also on that page are links to info about fixing some latency problems.

Now we are happy with the latency and must pick a servo period. In most cases a servo period of 1000000 ns is fine (that gives a 1 kHz servo calculation rate - 1000 calculations a second) if you are building a closed loop servo system that controls torque (current) rather than velocity (voltage) a faster rate would be better - something like 200000 (5 kHz calculation rate). The problem with lowering the servo rate is that it leaves less time available for the computer to do other things besides LinuxCNC's calculations. Typically the display (GUI) becomes less responsive. You must decide on a balance. Keep in mind that if you tune your closed loop servo system then change the servo period you probably will need to tune them again.

I/O Control Ports/Boards

PnCconf is capable of configuring machines that have up to two Mesa boards and three parallel ports. Parallel ports can only be used for simple low speed (servo rate) I/O.

Mesa

You must choose at least one Mesa board as PnCconf will not configure the parallel ports to count encoders or output step or PWM signals. The mesa cards available in the selection box are based on what PnCconf finds for firmware on the systems. There are options to add custom firmware and/or *blacklist* (ignore) some firmware or boards using a preference file. If no firmware is found PnCconf will show a warning and use internal sample firmware - no testing will be possible. One point to note is that if you choose two PCI Mesa cards there currently is no way to predict which card is 0 and which is 1 - you must test - moving the cards could change their order. If you configure with two cards both cards must be installed for tests to function.

Parallel Port

Up to 3 parallel ports (referred to as parports) can be used as simple I/O. You must set the address of the parport. You can either enter the Linux parallel port numbering system (0,1,or 2) or enter the actual address. The address for an on board parport is often 0x0278 or 0x0378 (written in hexadecimal) but can be found in the BIOS page. The BIOS page is found when you first start your computer you must press a key to enter it (such as F2). On the BIOS page you can find the parallel port address and set the mode such as SPP, EPP, etc on some computers this info is displayed for a few seconds during start up. For PCI parallel port cards the address can be found

by pressing the *parport address search* button. This pops up the help output page with a list of all the PCI devices that can be found. In there should be a reference to a parallel port device with a list of addresses. One of those addresses should work. Not all PCI parallel ports work properly. Either type can be selected as *in* (maximum amount of input pins) or *out* (maximum amount of output pins).

GUI Front-end list

This specifies the graphical display screens LinuxCNC will use. Each one has different option.

AXIS

- fully supports lathes.
- is the most developed and used front-end
- is designed to be used with mouse and keyboard
- is tkinter based so integrates PyVCP (Python based virtual control panels) naturally.
- has a 3D graphical window.
- allows VCP integrated on the side or in center tab

TkLinuxCNC

- hi contrast bright blue screen
- separate graphics window
- no VCP integration

TOUCHY

- Touchy was designed to be used with a touchscreen, some minimal physical switches and a MPG wheel.
- requires cycle-start, abort, and single-step signals and buttons
- It also requires shared axis MPG jogging to be selected.
- is GTK based so integrates GladeVCP (virtual control panels) naturally.
- allows VCP panels integrated in the center Tab
- has no graphical window
- look can be changed with custom themes

QtPlasmaC

- fully featured plasmac configuration based on the QtVCP infrastructure.
 - mouse/keyboard operation or touchscreen operation
 - no VCP integration
-

4.8.4 External Configuration

This page allows you to select external controls such as for jogging or overrides.

External Controls

☐ **USB Joystick Jogging**
Details

☐ **External Button Jogging**
Details

☒ **External MPG Jogging**
Details

☒ Shared MPG / selectable axis
☐ Mpg per axis
☒ selectable MPG increments
 increments

default	a)	b)	ab)	c)	ac)	bc)	abc)	d)	ad)	bd)	abcd)	in
0.0000	0.0001	0.0005	0.0010	0.0050	0.0100	0.0500	0.1000	0.0000	0.0000	0.0000	0.0000	in

Mux options

☒ use debounce 0.20 Sec
☒ use gray code
☐ ignore all inputs false

☐ **External Feed Override**
Details
☐ **Max Velocity Override**
Details
☐ **External Spindle Override**
Details

Help Cancel Back Forward

Figure 4.28: External Controls

If you select a Joystick for jogging, You will need it always connected for LinuxCNC to load. To use the analog sticks for useful jogging you probably need to add some custom HAL code. MPG jogging requires a pulse generator connected to a MESA encoder counter. Override controls can either use a pulse generator (MPG) or switches (such as a rotary dial). External buttons might be used with a switch based OEM joystick.

Joystick jogging

Requires a custom *device rule* to be installed in the system. This is a file that LinuxCNC uses to connect to Linux's device list. PnCconf will help to prepare this file.

- *Search for device rule* will search the system for rules, you can use this to find the name of devices you have already built with PnCconf.

- *Add a device rule* will allow you to configure a new device by following the prompts. You will need your device available.
- *test device* allows you to load a device, see its pin names and check its functions with halmeter.

joystick jogging uses HALUI and hal_input components.

External buttons

allows jogging the axis with simple buttons at a specified jog rate. Probably best for rapid jogging.

MPG Jogging

Allows you to use a Manual Pulse Generator to jog the machine's axis.

MPG's are often found on commercial grade machines. They output quadrature pulses that can be counted with a MESA encoder counter. PnCconf allows for an MPG per axis or one MPG shared with all axis. It allows for selection of jog speeds using switches or a single speed.

The selectable increments option uses the mux16 component. This component has options such as debounce and gray code to help filter the raw switch input.

Overrides

PnCconf allows overrides of feed rates and/or spindle speed using a pulse generator (MPG) or switches (eg. rotary).

4.8.5 GUI Configuration

Here you can set defaults for the display screens, add virtual control panels (VCP), and set some LinuxCNC options..

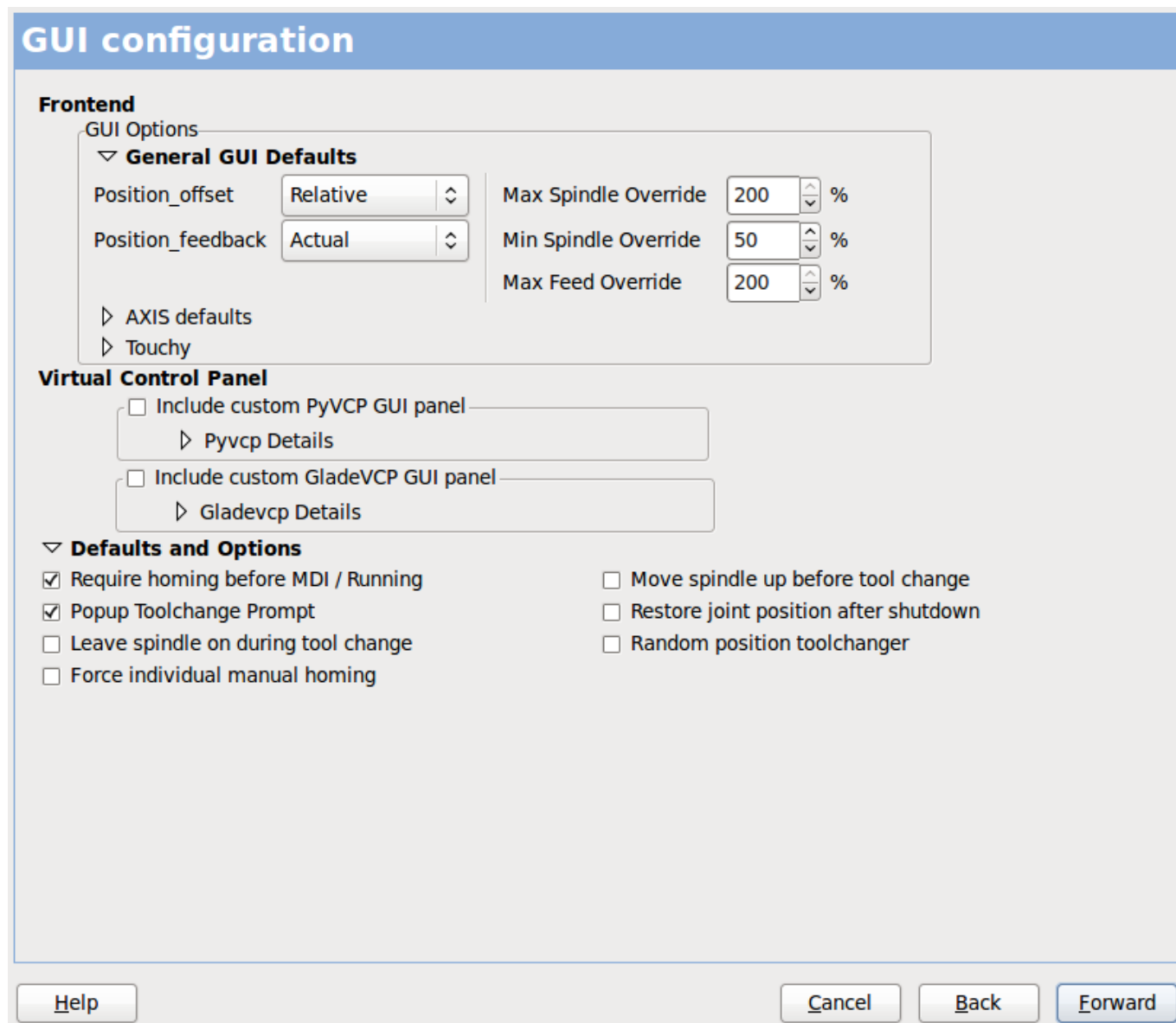


Figure 4.29: GUI Configuration

Front-end GUI Options

The default options allows general defaults to be chosen for any display screen.

AXIS defaults are options specific to AXIS. If you choose size , position or force maximize options then PnCconf will ask if it's alright to overwrite a preference file (.axisrc). Unless you have manually added commands to this file it is fine to allow it. Position and force max can be used to move AXIS to a second monitor if the system is capable.

Touchy defaults are options specific to Touchy. Most of Touchy's options can be changed while Touchy is running using the preference page. Touchy uses GTK to draw its screen, and GTK supports themes. Themes controls the basic look and feel of a program. You can download themes from the net or edit them yourself. There are a list of the current themes on the computer that you can pick from. To help some of the text to stand out PnCconf allows you to override the Themes's defaults. The position and force max options can be used to move Touchy to a second monitor if the system is capable.

QtPlasmaC options are specific to QtPlasmac, any common options that are not required will be disabled. If QtPlasmac is selected then the following screen will be a user button setup screen that is specific to QtPlasmaC and VCP options will not be available.

VCP options

Virtual Control Panels allow one to add custom controls and displays to the screen. AXIS and Touchy can integrate these controls inside the screen in designated positions. There are two kinds of VCPs - PyVCP which uses *Tkinter* to draw the screen and GladeVCP that uses *GTK* to draw the screen.

PyVCP

PyVCPs screen XML file can only be hand built. PyVCPs fit naturally in with AXIS as they both use *TKinter*.

HAL pins are created for the user to connect to inside their custom HAL file. There is a sample spindle display panel for the user to use as-is or build on. You may select a blank file that you can later add your controls *widgets* to or select a spindle display sample that will display spindle speed and indicate if the spindle is at requested speed.

PnCconf will connect the proper spindle display HAL pins for you. If you are using AXIS then the panel will be integrated on the right side. If not using AXIS then the panel will be separate *stand-alone* from the front-end screen.

You can use the geometry options to size and move the panel, for instance to move it to a second screen if the system is capable. If you press the *Display sample panel* button the size and placement options will be honored.

GladeVCP

GladeVCPs fit naturally inside of TOUCHY screen as they both use *GTK* to draw them, but by changing GladeVCP's theme it can be made to blend pretty well in AXIS (try Redmond).

It uses a graphical editor to build its XML files. HAL pins are created for the user to connect to, inside of their custom HAL file.

GladeVCP also allows much more sophisticated (and complicated) programming interaction, which PnCconf currently doesn't leverage (see GladeVCP in the manual)

PnCconf has sample panels for the user to use as-is or build on. With GladeVCP PnCconf will allow you to select different options on your sample display.

Under *sample options* select which ones you would like. The zero buttons use HALUI commands which you could edit later in the HALUI section.

Auto Z touch-off also requires the classic ladder touch-off program and a probe input selected. It requires a conductive touch-off plate and a grounded conductive tool. For an idea on how it works see:

http://wiki.linuxcnc.org/cgi-bin/wiki.pl?ClassicLadderExamples#Single_button_probe_touchoff

Under *Display Options*, size, position, and force max can be used on a *stand-alone* panel for such things as placing the screen on a second monitor if the system is capable.

You can select a *GTK* theme which sets the basic look and feel of the panel. You Usually want this to match the front-end screen. These options will be used if you press the *Display sample button*. With GladeVCP depending on the front-end screen, you can select where the panel will display.

You can force it to be stand-alone or with AXIS it can be in the center or on the right side, with Touchy it can be in the center.

Defaults and Options

- Require homing before MDI / Running
 - If you want to be able to move the machine before homing uncheck this checkbox.
- Popup Tool Prompt
 - Choose between an on screen prompt for tool changes or export standard signal names for a User supplied custom tool changer HAL file
- Leave spindle on during tool change:
 - Used for lathes
- Force individual manual homing
- Move spindle up before tool change
- Restore joint position after shutdown
 - Used for non-trivial kinematics machines
- Random position tool changers
 - Used for tool changers that do not return the tool to the same pocket. You will need to add custom HAL code to support tool changers.

4.8.6 Mesa Configuration

The Mesa configuration pages allow one to utilize different firmwares. On the basic page you selected a Mesa card here you pick the available firmware and select what and how many components are available.

Mesa0 Configuration-Board: 5i20 firmware: SVST8_4

Configuration Page

I/O Connector 2

I/O Connector 3

I/O Connector 4

Click on each page tab to configure signal names for each connector port.

The spin buttons below on this page allow you to select the amounts of different types of components. Press the button to make the tabbed pages accept the changes.

Board name: 5i20

Firmware: SVST8_4 ⬆ ⬇ ⬆

Mesa parport address: 0x378

PWM base frequency: 20000 ⬆ ⬇ ⬆ Hz

PDM base frequency: 6000 ⬆ ⬇ ⬆ Hz

Watchdog timeout: 10000000 ⬆ ⬇ ⬆ ns

Num of encoders: 4 ⬆ ⬇ ⬆

Num of pwm generators: 4 ⬆ ⬇ ⬆

Num of step generators: 3 ⬆ ⬇ ⬆

Num of GPIO: 42

Total number of pins: 72

Accept components Changes

Sanity Checks

☐ 7i29 daughter board

☐ 7i30 daughter board

☐ 7i33 daughter board

☐ 7i40 daughter board

Help
Cancel
Back
Forward

Figure 4.30: Mesa Board Configuration

Parport address is used only with Mesa parport card, the 7i43. An on board parallel port usually uses 0x278 or 0x378 though you should be able to find the address from the BIOS page. The 7i43 requires the parallel port to use the EPP mode, again set in the BIOS page. If using a PCI parallel port the address can be searched for by using the search button on the basic page.

Note

Many PCI cards do not support the EPP protocol properly.

PDM PWM and 3PWM base frequency sets the balance between ripple and linearity. If using Mesa daughter boards the docs for the board should give recommendations

**Important**

It's important to follow these to avoid damage and get the best performance.

The 7i33 requires PDM and a PDM base frequency of 6 MHz
The 7i29 requires PWM and a PWM base frequency of 20 kHz
The 7i30 requires PWM and a PWM base frequency of 20 kHz
The 7i40 requires PWM and a PWM base frequency of 50 kHz
The 7i48 requires UDM and a PWM base frequency of 24 kHz

Watchdog time out

is used to set how long the MESA board will wait before killing outputs if communication is interrupted from the computer. Please remember Mesa uses *active low* outputs meaning that when the output pin is on, it is low (approx 0 volts) and if it's off the output is high (approx 5 volts) make sure your equipment is safe when in the off (watchdog bitten) state.

Number of coders/PWM generators/STEP generators

You may choose the number of available components by deselecting unused ones. Not all component types are available with all firmware.

Choosing less than the maximum number of components allows one to gain more GPIO pins. If using daughter boards keep in mind you must not deselect pins that the card uses. For instance some firmware supports two 7i33 cards, If you only have one you may deselect enough components to utilize the connector that supported the second 7i33. Components are deselected numerically by the highest number first then down without skipping a number. If by doing this the components are not where you want them then you must use a different firmware. The firmware dictates where, what and the max amounts of the components. Custom firmware is possible, ask nicely when contacting the LinuxCNC developers and Mesa. Using custom firmware in PnCconf requires special procedures and is not always possible - Though I try to make PnCconf as flexible as possible.

After choosing all these options press the *Accept Component Changes* button and PnCconf will update the I/O setup pages. Only I/O tabs will be shown for available connectors, depending on the Mesa board.

4.8.7 Mesa I/O Setup

The tabs are used to configure the input and output pins of the Mesa boards. PnCconf allows one to create custom signal names for use in custom HAL files.

Mesa0 Configuration-Board: 5i20 firmware: SVST8_4

Configuration Page		I/O Connector 2		I/O Connector 3		I/O Connector 4	
Num	function	Pin Type	Inv	Num	function	Pin Type	Inv
	X Encoder	Quad Encoder-B	<input type="checkbox"/>		Multi Hand Wheel	Quad Encoder-B	<input type="checkbox"/>
1:	X Encoder	Quad Encoder-A	<input type="checkbox"/>	3:	Multi Hand Wheel	Quad Encoder-A	<input type="checkbox"/>
	Spindle Encoder	Quad Encoder-B	<input type="checkbox"/>		Unused Encoder	Quad Encoder-B	<input type="checkbox"/>
0:	Spindle Encoder	Quad Encoder-A	<input type="checkbox"/>	2:	Unused Encoder	Quad Encoder-A	<input type="checkbox"/>
	X Encoder	Quad Encoder-I	<input type="checkbox"/>		Multi Hand Wheel	Quad Encoder-I	<input type="checkbox"/>
	Spindle Encoder	Quad Encoder-I	<input type="checkbox"/>		Unused Encoder	Quad Encoder-I	<input type="checkbox"/>
1:	X Axis PWM	Pulse Width Gen-P	<input type="checkbox"/>	3:	Unused PWM Gen	Pulse Width Gen-P	<input type="checkbox"/>
0:	Spindle PWM	Pulse Width Gen-P	<input type="checkbox"/>	2:	Unused PWM Gen	Pulse Width Gen-P	<input type="checkbox"/>
	X Axis PWM	Pulse Width Gen-D	<input type="checkbox"/>		Unused PWM Gen	Pulse Width Gen-D	<input type="checkbox"/>
	Spindle PWM	Pulse Width Gen-D	<input type="checkbox"/>		Unused PWM Gen	Pulse Width Gen-D	<input type="checkbox"/>
	X Axis PWM	Pulse Width Gen-E	<input type="checkbox"/>		Unused PWM Gen	Pulse Width Gen-E	<input type="checkbox"/>
	Spindle PWM	Pulse Width Gen-E	<input type="checkbox"/>		Unused PWM Gen	Pulse Width Gen-E	<input type="checkbox"/>

Figure 4.31: Mesa I/O C2 Setup

On this tab with this firmware the components are setup for a 7i33 daughter board, usually used with closed loop servos. Note the component numbers of the encoder counters and PWM drivers are not in numerical order. This follows the daughter board requirements.

Mesa0 Configuration-Board: 5i20 firmware: SVST8_4

Configuration Page	I/O Connector 2	I/O Connector 3	I/O Connector 4		Num	function	Pin Type	Inv
					024:	X Minimum Limit + Hom	GPIO Input	<input type="checkbox"/>
					025:	X Maximum Limit	GPIO Input	<input type="checkbox"/>
					026:	Unused Input	GPIO Input	<input type="checkbox"/>
					027:	Unused Input	GPIO Input	<input type="checkbox"/>
					028:	Limits	GPIO Input	<input type="checkbox"/>
					029:	Home	GPIO Input	<input type="checkbox"/>
					030:	Limits/Home Shared	GPIO Input	<input type="checkbox"/>
					031:	Digital	GPIO Input	<input type="checkbox"/>
					032:	Axis Selection	GPIO Input	<input type="checkbox"/>
					033:	Overrides	GPIO Input	<input type="checkbox"/>
					034:	Spindle	GPIO Input	<input type="checkbox"/>
					035:	Operation	GPIO Input	<input type="checkbox"/>
						External Control	GPIO Input	<input type="checkbox"/>
						Axis rapid		
						X BLDC Control		
						Y BLDC Control		
						Z BLDC Control		
						A BLDC Control		
						S BLDC Control		
						Custom Signals		
						Launch test panel		
					036:	Jog incr A	GPIO Input	<input type="checkbox"/>
					037:	Jog incr B	GPIO Input	<input type="checkbox"/>
					038:	Jog incr C	GPIO Input	<input type="checkbox"/>
					039:	Joint select A	GPIO Input	<input type="checkbox"/>
					040:	Joint select B	GPIO Input	<input type="checkbox"/>
					041:	Spindle ON	GPIO Output	<input type="checkbox"/>
					042:	Spindle CW	GPIO Output	<input type="checkbox"/>
					043:	Spindle CCW	GPIO Output	<input type="checkbox"/>
					044:	Unused Output	GPIO Output	<input type="checkbox"/>
					045:	Coolant Flood	GPIO Output	<input type="checkbox"/>
					046:	Unused Output	GPIO Output	<input type="checkbox"/>
					047:	Unused Output	GPIO Output	<input type="checkbox"/>

Help
Cancel
Back
Forward

Figure 4.32: Mesa I/O C3 Setup

On this tab all the pins are GPIO. Note the 3 digit numbers - they will match the HAL pin number. GPIO pins can be selected as input or output and can be inverted.

Mesa0 Configuration-Board: 5i20 firmware: SVST8_4

Configuration Page	I/O Connector 2	I/O Connector 3	I/O Connector 4		Num	function	Pin Type	Inv
					0:	Y Axis StepGen	Step Gen-A	<input type="checkbox"/>
						Y Axis StepGen	Dir Gen-B	<input type="checkbox"/>
					050:	Unused Input	GPIO Input	<input type="checkbox"/>
					051:	Unused Input	GPIO Input	<input type="checkbox"/>
					052:	Unused Input	GPIO Input	<input type="checkbox"/>
					053:	Unused Input	GPIO Input	<input type="checkbox"/>
					1:	Z Axis StepGen	Step Gen-A	<input type="checkbox"/>
						Z Axis StepGen	Dir Gen-B	<input type="checkbox"/>
					056:	Unused Input	GPIO Input	<input type="checkbox"/>
					057:	Unused Input	GPIO Input	<input type="checkbox"/>
					058:	Unused Input	GPIO Input	<input type="checkbox"/>
					059:	Unused Input	GPIO Input	<input type="checkbox"/>
Launch test panel								

Num	function	Pin Type	Inv
2:	A Axis StepGen	Step Gen-A	<input type="checkbox"/>
	A Axis StepGen	Dir Gen-B	<input type="checkbox"/>
062:	Unused Input	GPIO Input	<input type="checkbox"/>
063:	Unused Input	GPIO Input	<input type="checkbox"/>
064:	Limits	GPIO Output	<input type="checkbox"/>
065:	Home	GPIO Output	<input type="checkbox"/>
066:	Limits/Home Shared	GPIO Output	<input type="checkbox"/>
067:	Digital	GPIO Output	<input type="checkbox"/>
068:	Axis Selection	GPIO Output	<input type="checkbox"/>
069:	Overrides	GPIO Output	<input type="checkbox"/>
070:	Spindle	Spindle	<input type="checkbox"/>
071:	Operation	Manual Spindle CW	<input type="checkbox"/>
	External Control	Manual Spindle CCW	<input type="checkbox"/>
	Axis rapid	Manual Spindle Stop	<input type="checkbox"/>
	X BLDC Control	Spindle Up-To-Speed	<input type="checkbox"/>
	Y BLDC Control		
	Z BLDC Control		
	A BLDC Control		
	S BLDC Control		
	Custom Signals		

[Help](#)
[Cancel](#)
[Back](#)
[Forward](#)

Figure 4.33: Mesa I/O C4 Setup

On this tab there are a mix of step generators and GPIO. Step generators output and direction pins can be inverted. Note that inverting a Step Gen-A pin (the step output pin) changes the step timing. It should match what your controller expects.

4.8.8 Parallel port configuration

First Parallel Port set for OUTPUT

Outputs (PC to Machine):		Invert	Inputs (Machine to PC):		Invert
Pin 1:	Digital out 0	<input type="checkbox"/>	Pin 2:	Unused Input	<input type="checkbox"/>
Pin 2:	Machine Is Enabled	<input type="checkbox"/>	Pin 3:	Unused Input	<input type="checkbox"/>
Pin 3:	X Amplifier Enable	<input type="checkbox"/>	Pin 4:	Unused Input	<input type="checkbox"/>
Pin 4:	Z Amplifier Enable	<input type="checkbox"/>	Pin 5:	Unused Input	<input type="checkbox"/>
Pin 5:	Unused Output	<input type="checkbox"/>	Pin 6:	Unused Input	<input type="checkbox"/>
Pin 6:	Unused Output	<input type="checkbox"/>	Pin 7:	Unused Input	<input type="checkbox"/>
Pin 7:	Unused Output	<input type="checkbox"/>	Pin 8:	Unused Input	<input type="checkbox"/>
Pin 8:	Unused Output	<input type="checkbox"/>	Pin 9:	Unused Input	<input type="checkbox"/>
Pin 9:	Unused Output	<input type="checkbox"/>	Pin 10:	Digital in 0	<input type="checkbox"/>
Pin 14:	Unused Output	<input type="checkbox"/>	Pin 11:	Unused Input	<input type="checkbox"/>
Pin 16:	Unused Output	<input type="checkbox"/>	Pin 12:	Unused Input	<input type="checkbox"/>
Pin 17:	Unused Output	<input type="checkbox"/>	Pin 13:	Unused Input	<input type="checkbox"/>
			Pin 15:	Unused Input	<input type="checkbox"/>

Launch Test Panel

Help

Cancel

Back

Forward

The parallel port can be used for simple I/O similar to Mesa's GPIO pins.

4.8.9 Axis Configuration

X Axis Motor/Encoder Configuration

Servo Info

P: 1.0000
 I: 0.0000
 D: 0.0000
 FF0: 0.0000
 FF1: 0.0000
 FF2: 0.0000
 Bias: 0.0000
 Deadband: 0.0000

Dac Output Scale: 10.00
 Dac Max Output: 10.00
 Dac Output Offset: 0.0000
 Quad Pulses / Rev: 4000

Stepper Info

Step On-Time: 1000
 Step Space: 1000
 Direction Hold: 1000
 Direction Setup: 1000
 Driver Type: Custom

☐ **Use Brushless Motor Control**

Details

Rapid Speed Following Error: 0.0050 inch
 Feed Speed Following Error: 0.0005 inch

☒ Invert Motor Direction
☐ Invert Encoder Direction

encoder Scale: 4000.000
 Stepper Scale: 0.000
 Maximum Velocity: 250 inch / min
 Maximum Acceleration: 2.0 inch / sec²

Open Loop Servo Test

Test / Tune Axis

Help Cancel Back Forward

Figure 4.34: Axis Drive Configuration

This page allows configuring and testing of the motor and/or encoder combination. If using a servo motor an open loop test is available, if using a stepper a tuning test is available.

Open Loop Test

An open loop test is important as it confirms the direction of the motor and encoder. The motor should move the axis in the positive direction when the positive button is pushed and also the encoder should count in the positive direction. The axis movement should follow the Machinery's Handbook⁴ standards or AXIS graphical display will not make much sense. Hopefully the help page and diagrams can help figure this out. Note that axis directions are based on TOOL movement not table movement. There is no acceleration ramping with the open loop test so start with lower DAC numbers. By moving the axis a known distance one can confirm the encoder scaling. The encoder should count even without the amp enabled depending on how power is supplied to the encoder.

⁴"axis nomenclature" in the chapter "Numerical Control" in the "Machinery's Handbook" published by Industrial Press.

**Warning**

If the motor and encoder do not agree on counting direction then the servo will run away when using PID control.

Since at the moment PID settings can not be tested in PnCconf the settings are really for when you re-edit a config - enter your tested PID settings.

DAC scale

DAC scaling, max output and offset are used to tailor the DAC output.

Compute DAC

These two values are the scale and offset factors for the axis output to the motor amplifiers. The second value (offset) is subtracted from the computed output (in volts), and divided by the first value (scale factor), before being written to the D/A converters. The units on the scale value are in true volts per DAC output volts. The units on the offset value are in volts. These can be used to linearize a DAC.

Specifically, when writing outputs, the LinuxCNC first converts the desired output in quasi-SI units to raw actuator values, e.g., volts for an amplifier DAC. This scaling looks like: The value for scale can be obtained analytically by doing a unit analysis, i.e., units are [output SI units]/[actuator units]. For example, on a machine with a velocity mode amplifier such that 1 volt results in 250 mm/sec velocity, Note that the units of the offset are in machine units, e.g., mm/sec, and they are pre-subtracted from the sensor readings. The value for this offset is obtained by finding the value of your output which yields 0.0 for the actuator output. If the DAC is linearized, this offset is normally 0.0.

The scale and offset can be used to linearize the DAC as well, resulting in values that reflect the combined effects of amplifier gain, DAC non-linearity, DAC units, etc. To do this, follow this procedure:

- Build a calibration table for the output, driving the DAC with a desired voltage and measuring the result:

Table 4.2: Output Voltage Measurements

Raw	Measured
-10	-9.93
-9	-8.83
0	-0.96
1	-0.03
9	9.87
10	10.07

- Do a least-squares linear fit to get coefficients a, b such that $meas = a * raw + b$
- Note that we want raw output such that our measured result is identical to the commanded output. This means
 - $cmd = a * raw + b$
 - $raw = (cmd - b) / a$
- As a result, the a and b coefficients from the linear fit can be used as the scale and offset for the controller directly.

MAX OUTPUT

The maximum value for the output of the PID compensation that is written to the motor amplifier, in volts. The computed output value is clamped to this limit. The limit is applied before scaling to raw output units. The value is applied symmetrically to both the plus and the minus side.

Tuning Test

The tuning test unfortunately only works with stepper based systems. Again confirm the directions on the axis is correct. Then test the system by running the axis back and forth, If the acceleration or max speed is too high you will lose steps. While jogging, Keep in mind it can take a while for an axis with low acceleration to stop. Limit switches are not functional during this test. You can set a pause time so each end of the test movement. This would allow you to set up and read a dial indicator to see if you are losing steps.

Stepper Timing

Stepper timing needs to be tailored to the step controller's requirements. PnCconf supplies some default controller timing or allows custom timing settings. See http://wiki.linuxcnc.org/cgi-bin/wiki.pl?Stepper_Drive_Timing for some more known timing numbers (feel free to add ones you have figured out). If in doubt use large numbers such as 5000 this will only limit max speed.

Brushless Motor Control

These options are used to allow low level control of brushless motors using special firmware and daughter boards. It also allows conversion of HALL sensors from one manufacturer to another. It is only partially supported and will require one to finish the HAL connections. Contact the mail-list or forum for more help.

Step Motor Scale	
<input checked="" type="checkbox"/> Pulley teeth (motor:Leadscrew):	1 : 2
<input type="checkbox"/> Worm turn ratio (Input:Output)	1 : 1
<input checked="" type="checkbox"/> Microstep Multiplication Factor:	5
<input type="checkbox"/> Leadscrew Metric Pitch	5.0000 mm / rev
<input checked="" type="checkbox"/> Leadscrew TPI	5.0000 TPI
Motor steps per revolution:	200

Encoder Scale	
<input type="checkbox"/> Pulley teeth (encoder:Leadscrew):	1 : 1
<input type="checkbox"/> Worm turn ratio (Input:Output)	1 : 1
<input type="checkbox"/> Leadscrew Metric Pitch	5.0000 mm / rev
<input type="checkbox"/> Leadscrew TPI	5.0000 TPI
Encoder lines per revolution:	1000 X 4 = Pulses/Rev

Calculated Scale	
motor steps per unit:	10000.0000
encoder pulses per unit:	4000.0000

Motion Data	
Calculated Axis SCALE:	10000.0 Steps / inch
Resolution:	0.0001000 inch / Step
Time to accelerate to max speed:	0.8335 sec
Distance to acheave max speed:	0.6947 inch
Pulse rate at max speed:	16.7 Khz
Motor RPM at max speed:	1000 RPM

Figure 4.35: Axis Scale Calculation

The scale settings can be directly entered or one can use the *calculate scale* button to assist. Use the check boxes to select appropriate calculations. Note that *pulley teeth* requires the number of teeth not the gear ratio. Worm turn ratio is just the opposite it requires the gear ratio. If your happy with the scale press apply otherwise push cancel and enter the scale directly.

X Axis Configuration

Positive Travel Distance (Machine zero Origin to end of + travel):		8.0
Negative Travel Distance (Machine zero Origin to end of - travel):		0.0
Home Position location (offset from machine zero Origin):		0.0
Home Switch location (Offset from machine zero Origin):		0.0
Home Search Velocity:	3	inch / min
Home Search Direction:	Towards Negative limit	
Home latch Velocity:	1	inch / min
Home Latch Direction:	Same	
Home Final Velocity:	0	inch / min
Use Encoder Index For Home:	NO	
<div style="display: flex; justify-content: space-between; align-items: flex-start;"> <div> <input type="checkbox"/> Use Compensation File: <div style="border: 1px solid black; padding: 2px; margin-left: 10px;">Type 1</div> </div> <div style="text-align: right;">filename: <div style="border: 1px solid black; padding: 2px; margin-left: 10px;">xcompensation</div></div> </div> <div style="display: flex; justify-content: space-between; align-items: flex-start; margin-top: 5px;"> <div> <input type="checkbox"/> Use Backlash Compensation: <div style="border: 1px solid black; padding: 2px; margin-left: 10px;">0.0000</div> </div> </div>		

Help

Cancel

Back

Forward

Figure 4.36: Axis Configuration

Also refer to the diagram tab for two examples of home and limit switches. These are two examples of many different ways to set homing and limits.

**Important**

It is very important to start with the axis moving in the right direction or else getting homing right is very difficult!

Remember positive and negative directions refer to the TOOL not the table as per the Machinists handbook.

On a typical knee or bed mill

- when the TABLE moves out that is the positive Y direction
- when the TABLE moves left that is the positive X direction
- when the TABLE moves down that is the positive Z direction

- when the HEAD moves up that is the positive Z direction

On a typical lathe

- when the TOOL moves right, away from the chuck
- that is the positive Z direction
- when the TOOL moves toward the operator
- that is the positive X direction. Some lathes have X opposite (e.g., tool on back side), that will work fine but AXIS graphical display can not be made to reflect this.

When using homing and / or limit switches LinuxCNC expects the HAL signals to be true when the switch is being pressed / tripped. If the signal is wrong for a limit switch then LinuxCNC will think the machine is on end of limit all the time. If the home switch search logic is wrong LinuxCNC will seem to home in the wrong direction. What it actually is doing is trying to BACK off the home switch.

Decide on limit switch location

Limit switches are the back up for software limits in case something electrical goes wrong, e.g., in case of a servo runaway. Limit switches should be placed so that the machine does not hit the physical end of the axis movement. Remember the axis will coast past the contact point if moving fast. Limit switches should be *active low* on the machine, i.e., power runs through the switches all the time - a loss of power (open switch) trips. While one could wire them the other way, this is fail safe. This may need to be inverted so that the HAL signal in LinuxCNC in *active high* - a TRUE means the switch was tripped. When starting LinuxCNC if you get an on-limit warning, and axis is NOT tripping the switch, inverting the signal is probably the solution. (use HALMETER to check the corresponding HAL signal eg. joint.0.pos-lim-sw-in X axis positive limit switch)

Decide on the home switch location

If you are using limit switches You may as well use one as a home switch. A separate home switch is useful if you have a long axis that in use is usually a long way from the limit switches or moving the axis to the ends presents problems of interference with material. Note, a long shaft in a lathe makes it hard to home to limits with out the tool hitting the shaft, so a separate home switch closer to the middle may be better. If you have an encoder with index then the home switch acts as a course home and the index will be the actual home location.

Decide on the MACHINE ORIGIN position

MACHINE ORIGIN is what LinuxCNC uses to reference all user coordinate systems from. I can think of little reason it would need to be in any particular spot. There are only a few G-codes that can access the MACHINE COORDINATE system.(G53, G30 and G28) If using tool-change-at-G30 option having the origin at the tool change position may be convenient. By convention, it may be easiest to have the ORIGIN at the home switch.

Decide on the (final) HOME POSITION

this just places the carriage at a consistent and convenient position after LinuxCNC figures out where the ORIGIN is.

Measure / calculate the positive / negative axis travel distances

Move the axis to the origin. Mark a reference on the movable slide and the non-movable support (so they are in line) move the machine to the end of limits. Measure between the marks that is one of the travel distances. Move the table to the other end of travel. Measure the marks again. That is the other travel distance. If the ORIGIN is at one of the limits then that travel distance will be zero.

(machine) ORIGIN

The Origin is the MACHINE zero point. (not the zero point you set your cutter / material at). LinuxCNC uses this point to reference everything else from. It should be inside the software limits. LinuxCNC uses the home switch location to calculate the origin position (when using home switches or must be manually set if not using home switches).

Travel distance

This is the maximum distance the axis can travel in each direction. This may or may not be able to be measured directly from origin to limit switch. The positive and negative travel distances should add up to the total travel distance.

POSITIVE TRAVEL DISTANCE

This is the distance the Axis travels from the Origin to the positive travel distance or the total travel minus the negative travel distance. You would set this to zero if the origin is positioned at the positive limit. This will always be zero or a positive number.

NEGATIVE TRAVEL DISTANCE

This is the distance the Axis travels from the Origin to the negative travel distance, or the total travel minus the positive travel distance. You would set this to zero if the origin is positioned at the negative limit. This will always be zero or a negative number. If you forget to make this negative PnCconf will do it internally.

(Final) HOME POSITION

This is the position the home sequence will finish at. It is referenced from the Origin so can be negative or positive depending on what side of the Origin it is located. When at the (final) home position if you must move in the Positive direction to get to the Origin, then the number will be negative.

HOME SWITCH LOCATION

This is the distance from the home switch to the Origin. It could be negative or positive depending on what side of the Origin it is located. When at the home switch location if you must move in the Positive direction to get to the Origin, then the number will be negative. If you set this to zero then the Origin will be at the location of the limit switch (plus distance to find index if used).

Home Search Velocity

Course home search velocity in units per minute.

Home Search Direction

Sets the home switch search direction either negative (i.e., towards negative limit switch) or positive (i.e., towards positive limit switch).

Home Latch Velocity

Fine Home search velocity in units per minute.

Home Final Velocity

Velocity used from latch position to (final) home position in units per minute. Set to 0 for max rapid speed.

Home latch Direction

Allows setting of the latch direction to the same or opposite of the search direction.

Use Encoder Index For Home

LinuxCNC will search for an encoder index pulse while in the latch stage of homing.

Use Compensation File

Allows specifying a Comp filename and type. Allows sophisticated compensation. See [AXIS Section](#) of the INI chapter.

Use Backlash Compensation

Allows setting of simple backlash compensation. Can not be used with Compensation File. See [AXIS Section](#) of the INI chapter.

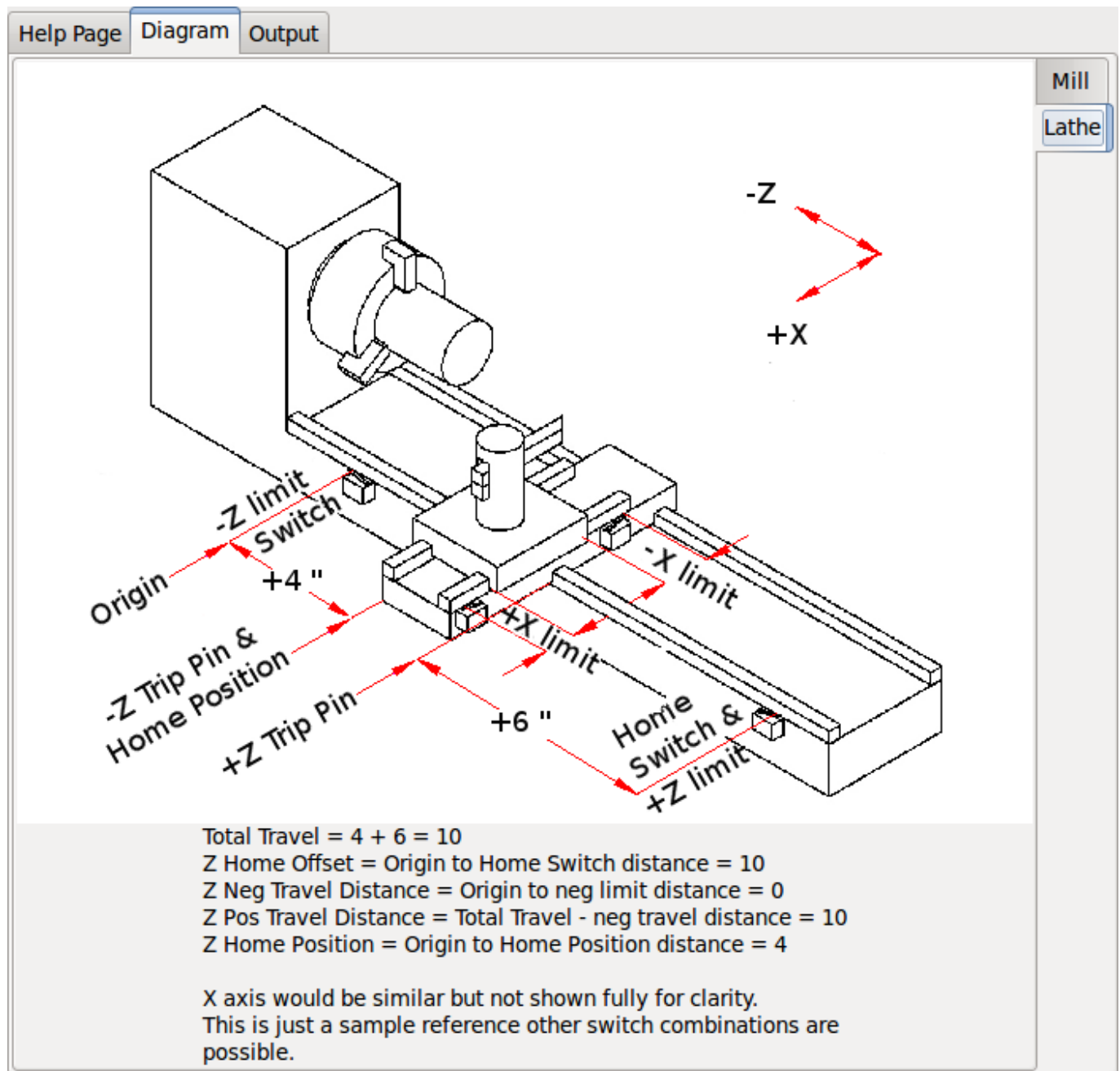


Figure 4.37: AXIS Help Diagram

The diagram should help to demonstrate an example of limit switches and standard axis movement directions. In this example the Z axis was two limit switches, the positive switch is shared as a home switch. The MACHINE ORIGIN (zero point) is located at the negative limit. The left edge of the carriage is the negative trip pin and the right the positive trip pin. We wish the FINAL HOME POSITION to be 4 inches away from the ORIGIN on the positive side. If the carriage was moved to the positive limit we would measure 10 inches between the negative limit and the negative trip pin.

4.8.10 Spindle Configuration

If you select spindle signals then this page is available to configure spindle control.

Tip

Many of the option on this page will not show unless the proper option was selected on previous pages!

Spindle Motor/Encoder Configuration

Servo Info
P 1.0000
I 0.0000
D 0.0000
FF0 0.0000
FF1 0.0000
FF2 0.0000
Bias 0.0000
Deadband 0.0000

Dac Output Scale: 10.00
Dac Max Output: 10.00
Dac Output Offset: 0.0000
Quad Pulses / Rev: 4000
Open Loop Servo Test

Stepper Info
Step On-Time 1000
Step Space 1000
Direction Hold 1000
Direction Setup 1000
Driver Type: Custom

☐ **Use Brushless Motor Control**
Details

☐ **Use Spindle-At-Speed**
Scale: 95 %

Rapid Speed Following Error: 0.0000 rev
Feed Speed Following Error: 0.0000 rev
☐ Invert Motor Direction
☐ Invert Encoder Direction
Test / Tune Axis

encoder Scale: 4000.000
Stepper Scale: 0.000
Maximum Velocity: 100 rev / min
Maximum Acceleration: 2.0 rev / sec²
Calculate Scale

Help Cancel Back Forward

Figure 4.38: Spindle Motor/Encoder Configuration

This page is similar to the axis motor configuration page.

There are some differences:

- Unless one has chosen a stepper driven spindle there is no acceleration or velocity limiting.
- There is no support for gear changes or ranges.
- If you picked a VCP spindle display option then spindle-at-speed scale and filter settings may be shown.
- Spindle-at-speed allows LinuxCNC to wait till the spindle is at the requested speed before moving the axis. This is particularly handy on lathes with constant surface feed and large speed diameter changes. It requires either encoder feedback or a digital spindle-at-speed signal typically connected to a VFD drive.

- If using encoder feedback, you may select a spindle-at-speed scale setting that specifies how close the actual speed must be to the requested speed to be considered at-speed.
- If using encoder feedback, the VCP speed display can be erratic - the filter setting can be used to smooth out the display. The encoder scale must be set for the encoder count / gearing used.
- If you are using a single input for a spindle encoder you must add the line: `setp hm2_7i43.0.encoder.00.count mode 1` (changing the board name and encoder number to your requirements) into a custom HAL file. See the [Encoders Section](#) in Hostmot2 for more info about counter mode.

4.8.11 Advanced Options

This allows setting of HALUI commands and loading of ClassicLadder and sample ladder programs. If you selected GladeVCP options such as for zeroing axis, there will be commands showing. See the [HALUI Chapter](#) for more info on using custom halcmds. There are several ladder program options. The Estop program allows an external ESTOP switch or the GUI frontend to throw an Estop. It also has a timed lube pump signal. The Z auto touch-off is with a touch-off plate, the GladeVCP touch-off button and special HALUI commands to set the current user origin to zero and rapid clear. The serial modbus program is basically a blank template program that sets up ClassicLadder for serial modbus. See the << cha:classicladder,Classicladder Chapter >> in the manual.

Advanced Options

☒ Include Halui user interface component / commands

Cmd 1G10 L20 P0 XO

Cmd 2

Cmd 3

Cmd 4

Cmd 5

Cmd 6

Cmd 7

Cmd 8

Cmd 9

Cmd 10

Cmd 11

Cmd 12

Cmd 13

Cmd 14

Cmd 15

☒ Include Classicladder PLC

▼ Setup number of external pins

Number of digital (bit) in pins:15

Number of digital (bit) out pins:15

Number of analog (s32) in pins:10

Number of analog (s32) out pins:10

Number of analog (float) in pins:10

Number of analog (float) out pins:10

☐ Include modbus master support

☐ Blank ladder program

☒ Estop ladder program

☐ Z Auto Touch off program

☐ Serial modbus program

☐ Existing custom program

☒ Include connections to HAL

Edit ladder program

Help

Cancel

Back

Forward

Figure 4.39: PnCConf, advanced options

4.8.12 HAL Components

On this page you can add additional HAL components you might need for custom HAL files. In this way one should not have to hand edit the main HAL file, while still allowing user needed components.

HAL Component Page

Add HAL components with this page.

Component number of components

Absolute

PID

scale

mux16

▼ **Custom Components Commands**

Load Command	Thread Command
loadrt example_comp	addf example_comp_calcs

Thread Speed

Servo Thread

Base Thread

Help Cancel Back Forward

Figure 4.40: HAL Components

The first selection is components that pncconf uses internally. You may configure pncconf to load extra instances of the components for your custom HAL file.

Select the number of instances your custom file will need, PnCconf will add what it needs after them. Meaning if you need 2 and PnCconf needs 1 PnCconf will load 3 instances and use the last one.

Custom Component Commands

This selection will allow you to load HAL components that PnCconf does not use. Add the loadrt or loadusr command, under the heading *loading command*. Add the addf command under the heading *Thread command*. The components will be added to the thread between reading of inputs and writing of outputs, in the order you write them in the *thread command*.

4.8.13 Advanced Usage Of PnCconf

PnCconf does its best to allow flexible customization by the user. PnCconf has support for custom signal names, custom loading of components, custom HAL files and custom firmware.

There are also signal names that PnCconf always provides regardless of options selected, for user's custom HAL files. With some thought most customizations should work regardless if you later select different options in PnCconf.

Eventually if the customizations are beyond the scope of PnCconf's frame work you can use PnCconf to build a base config or use one of LinuxCNC's sample configurations and just hand edit it to what ever you want.

Custom Signal Names

If you wish to connect a component to something in a custom HAL file write a unique signal name in the combo entry box. Certain components will add endings to your custom signal name:

Encoders will add < customname > +:

- position
- count
- velocity
- index-enable
- reset

Steppers add:

- enable
- counts
- position-cmd
- position-fb
- velocity-fb

PWM add:

- enable
- value

GPIO pins will just have the entered signal name connected to it

In this way one can connect to these signals in the custom HAL files and still have the option to move them around later.

Custom Signal Names

The HAL Components page can be used to load components needed by a user for customization.

Loading Custom Firmware

PnCconf searches for firmware on the system and then looks for the XML file that it can convert to what it understands. These XML files are only supplied for officially released firmware from the LinuxCNC team. To utilize custom firmware one must convert it to an array that PnCconf understands and add its file path to PnCconf's preference file. By default this path searches the desktop for a folder named custom_firmware and a file named firmware.py.

The hidden preference file is in the user's home file, is named `.pncconf-preferences` and require one to select *show hidden files* in your file manager to see and edit it or on the command line you use `ls` with the `-a` option. The contents of this file can be seen when you first load PnCconf - press the help button and look at the output page.

Ask on the LinuxCNC mail-list or forum for info about converting custom firmware. Not all firmware can be utilized with PnCconf.

Custom HAL Files

There are four custom files that you can use to add HAL commands to:

- `custom.hal` is for HAL commands that don't have to be run after the GUI frontend loads. It is run after the configuration-named HAL file.
- `custom_postgui.hal` is for commands that must be run after AXIS loads or a standalone PyVCP display loads.
- `custom_gvcp.hal` is for commands that must be run after GladeVCP is loaded.
- `shutdown.hal` is for commands to run when LinuxCNC shuts down in a controlled manner.

4.9 Linux FAQ

These are some basic Linux commands and techniques for new to Linux users. More complete information can be found on the web or by using the man pages.

4.9.1 Automatic Login

4.9.1.1 Debian

Debian Stretch uses the Xfce desktop environment by default, with the lightDM display manager lightDM. To get automatic login with Stretch:

- In a terminal, use the command:

```
$ /usr/sbin/lightdm --show-config
```

- Make a note of the absolute path to the configuration file `lightdm.conf`.
- Edit that file with a pure text editor (`gedit`, `nano`, etc), as root.
- Find and uncomment the lines:

```
#autologin-user=  
#autologin-user-timeout=0
```

- Set `autologin-user=your_user_name`
- Save and reboot.

4.9.1.2 Ubuntu

When you install LinuxCNC with the Ubuntu LiveCD the default is to have to log in each time you turn the computer on. To enable automatic login go to *System > Administration > Login Window*. If it is a fresh install the Login Window might take a second or three to pop up. You will have to have your password that you used for the install to gain access to the Login Window Preferences window. In the Security tab check off Enable Automatic Login and pick a user name from the list (that would be you).

4.9.2 Automatic Startup

To have LinuxCNC start automatically with your config after turning on the computer go to *System > Preferences > Sessions > Startup Applications*, click Add. Browse to your config and select the .ini file. When the file picker dialog closes, add linuxcnc and a space in front of the path to your .ini file.

Example:

```
linuxcnc /home/mill/linuxcnc/config/mill/mill.ini
```

The documentation refers to your respective .ini file as INI-file.

4.9.3 Terminal

Many things need to be done from the terminal like checking the kernel message buffer with *dmesg*. Ubuntu and Linux Mint have a keyboard shortcut Ctrl + Alt + t. Debian Stretch does not have any keyboard shortcuts defined. It can be easily created with the *Configuration Manager*. Most modern file managers support the right key to open a terminal just make sure your right clicking on a blank area or a directory not a file name. Most OS's have the terminal as a menu item, usually in Accessories.

4.9.4 Man Pages

A man page (short for manual page) is a form of software documentation usually found on a Unix or Unix-like operating system like Linux.

To view a man page open up a terminal to find out something about the find command in the terminal window type:

```
man find
```

Use the Page Up and Page Down keys to view the man page and the Q key to quit viewing.

Note

Viewing the man page from the terminal may not get the expected man page. For example if you type in man abs you will get the C abs not the LinuxCNC abs. It is best to view the LinuxCNC man pages in the HTML documents.

4.9.5 List Modules

Sometimes when troubleshooting you need to get a list of modules that are loaded. In a terminal window type:

```
lsmod
```

If you want to send the output from lsmod to a text file in a terminal window type:

```
lsmod > mymod.txt
```

The resulting text file will be located in the home directory if you did not change directories when you opened up the terminal window and it will be named mymod.txt or what ever you named it.

4.9.6 Editing a Root File

When you open the file browser and you see the owner of the file is root you must do extra steps to edit that file. Editing some root files can have bad results. Be careful when editing root files. Generally, you can open and view most root files, but they will open in *read only* mode.

4.9.6.1 The Command Line Way

Open a terminal and type

```
sudo gedit
```

Open the file with File > Open > Edit

4.9.6.2 The GUI Way

1. Right click on the desktop and select Create Launcher
2. Type a name in like sudo edit
3. Type *gksudo "gnome-open %u"* as the command and save the launcher to your desktop
4. Drag a file onto your launcher to open and edit

4.9.6.3 Root Access

In Ubuntu you can become root by typing in "sudo -i" in a terminal window then typing in your password. Be careful, because you can really foul things up as root if you don't know what you're doing.

4.9.7 Terminal Commands

4.9.7.1 Working Directory

To find out the path to the present working directory in the terminal window type:

```
pwd
```

4.9.7.2 Changing Directories

To change the working directory to the one one level up, i.e., the parent directory, in the terminal window type:

```
cd ..
```

To move up two levels in the terminal window type:

```
cd ../../
```

To move directly to your home directory, in the terminal window use the cd command with no arguments:

```
cd
```

To move down to the linuxcnc/configs subdirectory in the terminal window type:

```
cd linuxcnc/configs
```

4.9.7.3 Listing files in a directory

To view a list of all the files and subdirectories in the terminal window type:

```
dir
```

or

```
ls
```

4.9.7.4 Finding a File

The find command can be a bit confusing to a new Linux user. The basic syntax is:

```
find starting-directory parameters actions
```

For example to find all the .ini files in your linuxcnc directory you first need to use the pwd command to find out the directory.

Open a new terminal window and type:

```
pwd
```

And pwd might return the following result:

```
/home/joe
```

With this information put the command together like this:

```
find /home/joe/linuxcnc -name \*.ini -print
```

The -name is the name of the file your looking for and the -print tells it to print out the result to the terminal window. The *.ini tells find to return all files that have the .ini extension. The backslash is needed to escape the shell meta-characters. See the find man page for more information on find.

4.9.7.5 Searching for Text

```
grep -irl 'text to search for' *
```

This will find all the files that contain the *text to search for* in the current directory and all the subdirectories below it, while ignoring the case. The -i is for ignore case and the -r is for recursive (include all subdirectories in the search). The -l option will return a list of the file names, if you leave the -l off you will also get the text where each occurrence of the "text to search for" is found. The * is a wild card for search all files. See the grep man page for more information.

4.9.7.6 Diagnostic Messages

To view the diagnostic messages use "dmesg" from the command window. To save the diagnostic messages to a file use the redirection operator >, like this:

```
dmesg > bootmsg.txt
```

The contents of this file can be copied and pasted on line to share with people trying to help you diagnose your problem.

To clear the message buffer type this:

```
sudo dmesg -c
```

This can be helpful to do just before launching LinuxCNC, so that there will only be a record of information related to the current launch of LinuxCNC.

To find the built in parallel port address use grep to filter the information out of dmesg.

After boot up open a terminal and type:

```
dmesg|grep parport
```

4.9.8 Convenience Items

4.9.8.1 Terminal Launcher

If you want to add a terminal launcher to the panel bar on top of the screen you typically can right click on the panel at the top of the screen and select "Add to Panel". Select Custom Application Launcher and Add. Give it a name and put `gnome-terminal` in the command box.

4.9.9 Hardware Problems

4.9.9.1 Hardware Info

To find out what hardware is connected to your motherboard in a terminal window type:

```
lspci -v
```

4.9.9.2 Monitor Resolution

During installation Ubuntu attempts to detect the monitor settings. If this fails you are left with a generic monitor with a maximum resolution of 800x600.

Instructions for fixing this are located here:

<https://help.ubuntu.com/community/FixVideoResolutionHowto>

4.9.10 Paths

Relative Paths Relative paths are based on the startup directory which is the directory containing the INI-file. Using relative paths can facilitate relocation of configurations but requires a good understanding of linux path specifiers.

```
./f0      is the same as f0, e.g., a file named f0 in the startup directory
../f1     refers to a file f1 in the parent directory
../../f2  refers to a file f2 in the parent of the parent directory
../.../f3 etc.
```

4.10 Lathe User Information

This chapter will provide information specific to lathes.

4.10.1 Lathe Mode

If your CNC machine is a lathe, there are some specific changes you will probably want to make to your INI file in order to get the best results from LinuxCNC.

If you are using the AXIS display, have AXIS display your lathe tools properly. See the [INI Configuration](#) section for more details.

To set up AXIS for Lathe Mode.

```
[DISPLAY]

# Tell the AXIS GUI our machine is a lathe.
LATHE = TRUE
```

Lathe Mode in AXIS does not set your default plane to G18 (XZ). You must program that in the preamble of each G-code file or (better) add it to your INI file, like this:

```
[RS274NGC]

# G-code modal codes (modes) that the interpreter is initialized with
# on startup
RS274NGC_STARTUP_CODE = G18 G20 G90
```

If your using Gmoccapy then see the [the Gmoccapy Lathe](#) section.

4.10.2 Lathe Tool Table

The "Tool Table" is a text file that contains information about each tool. The file is located in the same directory as your configuration and is called "tool.tbl" by default. The tools might be in a tool changer or just changed manually. The file can be edited with a text editor or be updated using G10 L1,L10,L11. There is also a built-in tool table editor in the AXIS display. The maximum number of entries in the tool table is 56. The maximum tool and pocket number is 99999.

Earlier versions of LinuxCNC had two different tool table formats for mills and lathes, but since the 2.4.x release, one tool table format is used for all machines. Just ignore the parts of the tool table that don't pertain to your machine, or which you don't need to use. For more information on the specifics of the tool table format, see the [Tool Table](#) Section.

4.10.3 Lathe Tool Orientation

The following figure shows the lathe tool orientations with the center line angle of each orientation and info on FRONTANGLE and BACKANGLE.

The FRONTANGLE and BACKANGLE are clockwise starting at a line parallel to Z+.

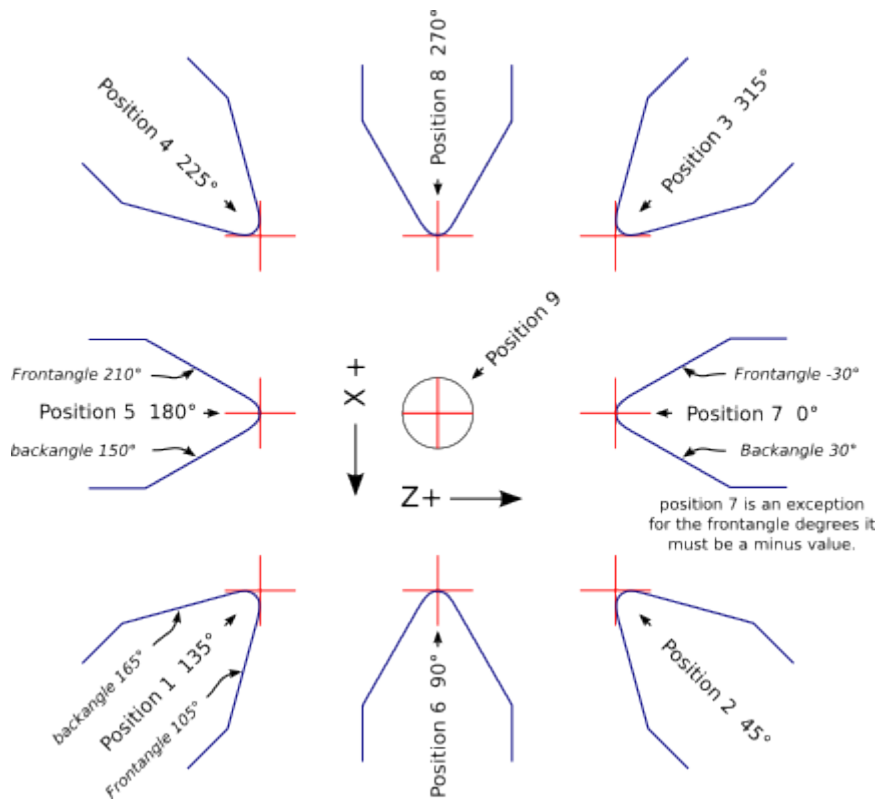
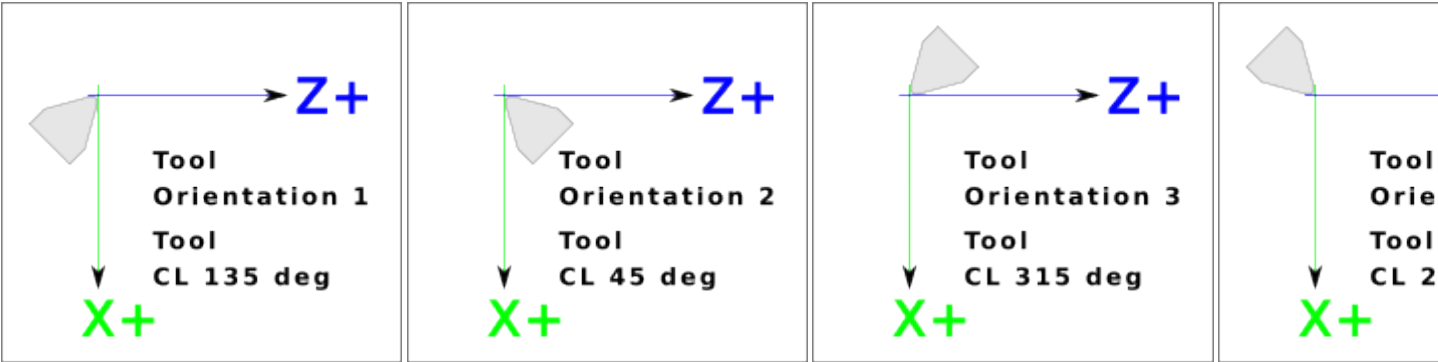


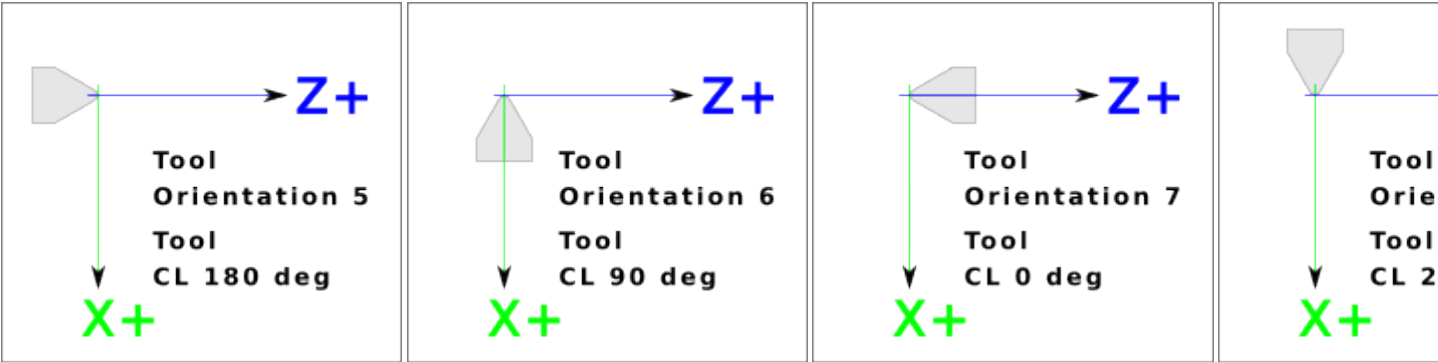
Figure 4.41: Lathe Tool Orientations

In AXIS the following figures show what the Tool Positions look like, as entered in the tool table.

Tool Positions 1, 2, 3 & 4
Tool Positions 123 & 4 23 & 4 3 & 4



Tool Positions 5, 6, 7 & 8
Tool Positions 567 & 8 67 & 8 7 & 8



4.10.4 Tool Touch Off

When running in lathe mode in AXIS you can set the X and Z in the tool table using the Touch Off window. If you have a tool turret you normally have *Touch off to fixture* selected when setting up your turret. When setting the material Z zero you have *Touch off to material* selected. For more information on the G-codes used for tools see [M6](#), [Tn](#), and [G43](#). For more information on tool touch off options in AXIS see [Tool Touch Off](#).

4.10.4.1 X Touch Off

The X axis offset for each tool is normally an offset from the center line of the spindle.

One method is to take your normal turning tool and turn down some stock to a known diameter. Using the Tool Touch Off window enter the measured diameter (or radius if in radius mode) for that tool. Then using some layout fluid or a marker to coat the part bring each tool up till it just touches the dye and set it's X offset to the diameter of the part used using the tool touch off. Make sure any tools in the corner quadrants have the nose radius set properly in the tool table so the control point is correct. Tool touch off automatically adds a G43 so the current tool is the current offset.

A typical session might be:

1. Home each axis if not homed.
2. Set the current tool with *Tn M6 G43* where *n* is the tool number.
3. Select the X axis in the *Manual Control window*.
4. Move the X to a known position or take a test cut and measure the diameter.
5. Select Touch Off and pick Tool Table then enter the position or the diameter.
6. Follow the same sequence to correct the Z axis.

Note: if you are in Radius Mode you must enter the radius, not the diameter.

4.10.4.2 Z Touch Off

The Z axis offsets can be a bit confusing at first because there are two elements to the Z offset. There is the tool table offset, and the machine coordinate offset. First we will look at the tool table offsets. One method is to use a fixed point on your lathe and set the Z offset for all tools from this point. Some use the spindle nose or chuck face. This gives you the ability to change to a new tool and set its Z offset without having to reset all the tools.

A typical session might be:

1. Home each axis if not homed.
 2. Make sure no offsets are in effect for the current coordinate system.
 3. Set the current tool with *Tn M6 G43* where *n* is the tool number.
 4. Select the Z axis in the Manual Control window.
 5. Bring the tool close to the control surface.
 6. Using a cylinder move the Z away from the control surface until the cylinder just passes between the tool and the control surface.
 7. Select Touch Off and pick Tool Table and set the position to 0.0.
-

8. Repeat for each tool using the same cylinder.

Now all the tools are offset the same distance from a standard position. If you change a tool like a drill bit you repeat the above and it is now in sync with the rest of the tools for Z offset. Some tools might require a bit of cyphering to determine the control point from the touch off point. For example, if you have a 0.125" wide parting tool and you touch the left side off but want the right to be Z0, then enter 0.125" in the touch off window.

4.10.4.3 The Z Machine Offset

Once all the tools have the Z offset entered into the tool table, you can use any tool to set the machine offset using the machine coordinate system.

A typical session might be:

1. Home each axis if not homed.
2. Set the current tool with *Tn M6* where *n* is the tool number.
3. Issue a G43 so the current tool offset is in effect.
4. Bring the tool to the work piece and set the machine Z offset.

If you forget to set the G43 for the current tool when you set the machine coordinate system offset, you will not get what you expect, as the tool offset will be added to the current offset when the tool is used in your program.

4.10.5 Spindle Synchronized Motion

Spindle synchronized motion requires a quadrature encoder connected to the spindle with one index pulse per revolution. See the motion man page and the [Spindle Control Example](#) for more information.

Threading The G76 threading cycle is used for both internal and external threads. For more information see the [G76](#) Section.

Constant Surface Speed CSS or Constant Surface Speed uses the machine X origin modified by the tool X offset to compute the spindle speed in RPM. CSS will track changes in tool offsets. The [X machine origin](#) should be when the reference tool (the one with zero offset) is at the center of rotation. For more information see the [G96](#) Section.

Feed per Revolution Feed per revolution will move the Z axis by the F amount per revolution. This is not for threading, use G76 for threading. For more information see the [G95](#) Section.

4.10.6 Arcs

Calculating arcs can be mind challenging enough without considering radius and diameter mode on lathes as well as machine coordinate system orientation. The following applies to center format arcs. On a lathe you should include G18 in your preamble as the default is G17 even if you're in lathe mode, in the user interface AXIS. Arcs in G18 XZ plane use I (X axis) and K (Z axis) offsets.

4.10.6.1 Arcs and Lathe Design

The typical lathe has the spindle on the left of the operator and the tools on the operator side of the spindle center line. This is typically set up with the imaginary Y axis (+) pointing at the floor.

The following will be true on this type of setup:

- The Z axis (+) points to the right, away from the spindle.
- The X axis (+) points toward the operator, and when on the operator side of the spindle the X values are positive.

Some lathes with tools on the back side have the imaginary Y axis (+) pointing up.

G2/G3 Arc directions are based on the axis they rotate around. In the case of lathes, it is the imaginary Y axis. If the Y axis (+) points toward the floor, you have to look up for the arc to appear to go in the correct direction. So looking from above you reverse the G2/G3 for the arc to appear to go in the correct direction.

4.10.6.2 Radius & Diameter Mode

When calculating arcs in radius mode you only have to remember the direction of rotation as it applies to your lathe.

When calculating arcs in diameter mode X is diameter and the X offset (I) is radius even if you're in G7 diameter mode.

4.10.7 Tool Path

4.10.7.1 Control point

The control point for the tool follows the programmed path. The control point is the intersection of a line parallel to the X and Z axis and tangent to the tool tip diameter, as defined when you touch off the X and Z axes for that tool. When turning or facing straight sided parts the cutting path and the tool edge follow the same path. When turning radius and angles the edge of the tool tip will not follow the programmed path unless cutter comp is in effect. In the following figures you can see how the control point does not follow the tool edge as you might assume.

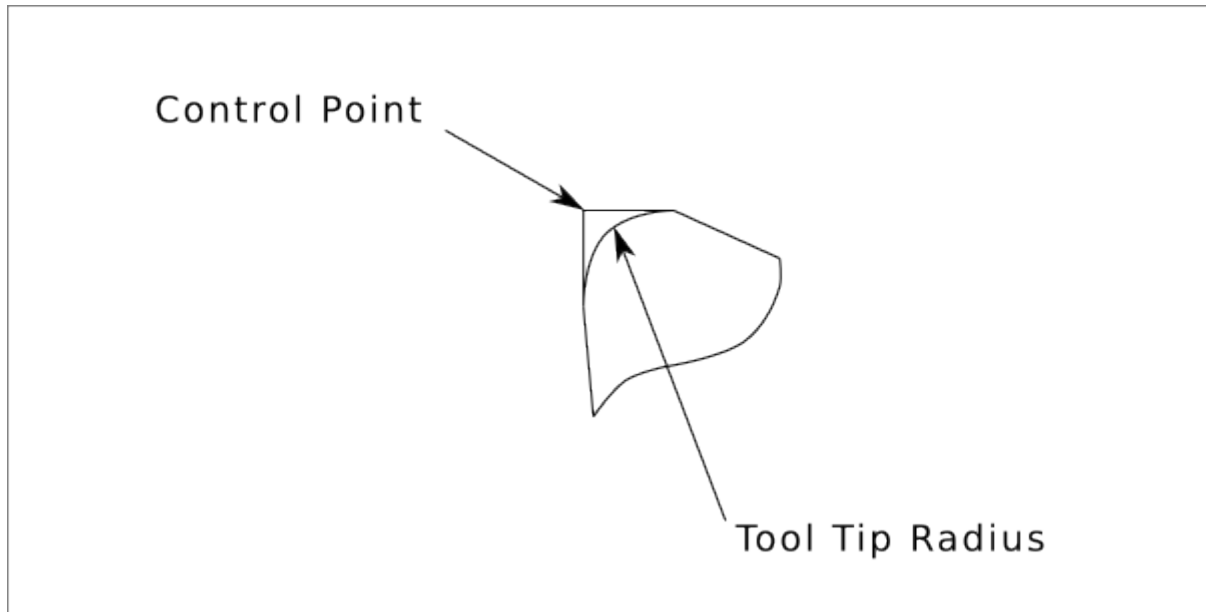


Figure 4.42: Control point

4.10.7.2 Cutting Angles without Cutter Comp

Now imagine we program a ramp without cutter comp. The programmed path is shown in the following figure. As you can see in the figure the programmed path and the desired cut path are one and the same as long as we are moving in an X or Z direction only.

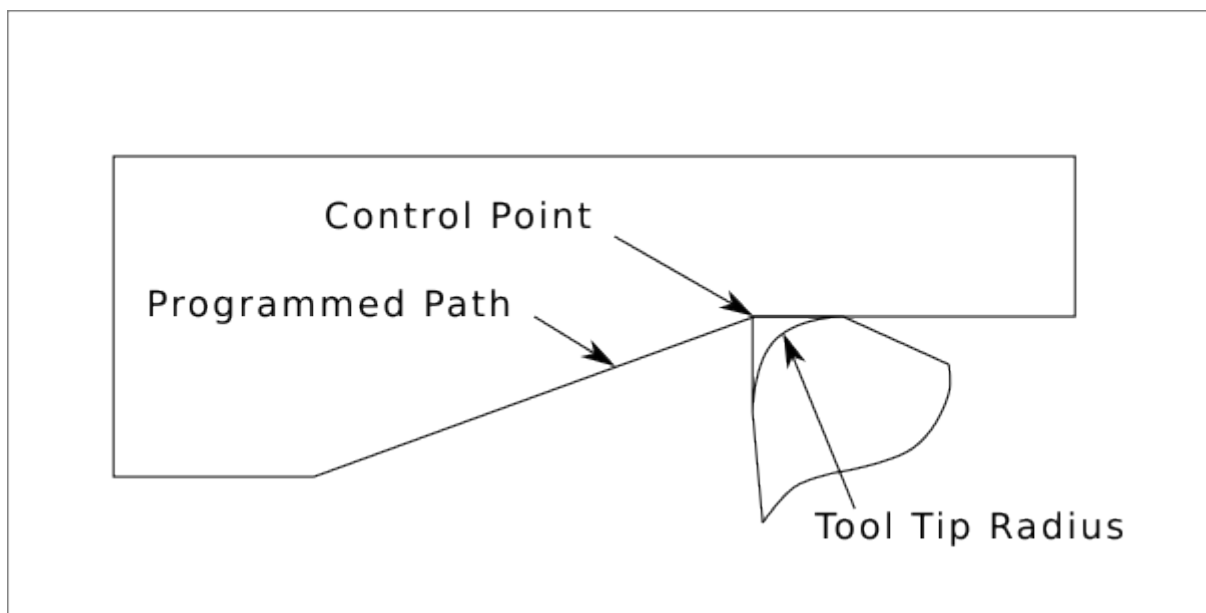


Figure 4.43: Ramp Entry

Now as the control point progresses along the programmed path the actual cutter edge does not follow the programmed path as shown in the following figure. There are two ways to solve this, cutter comp and adjusting your programmed path to compensate for tip radius.

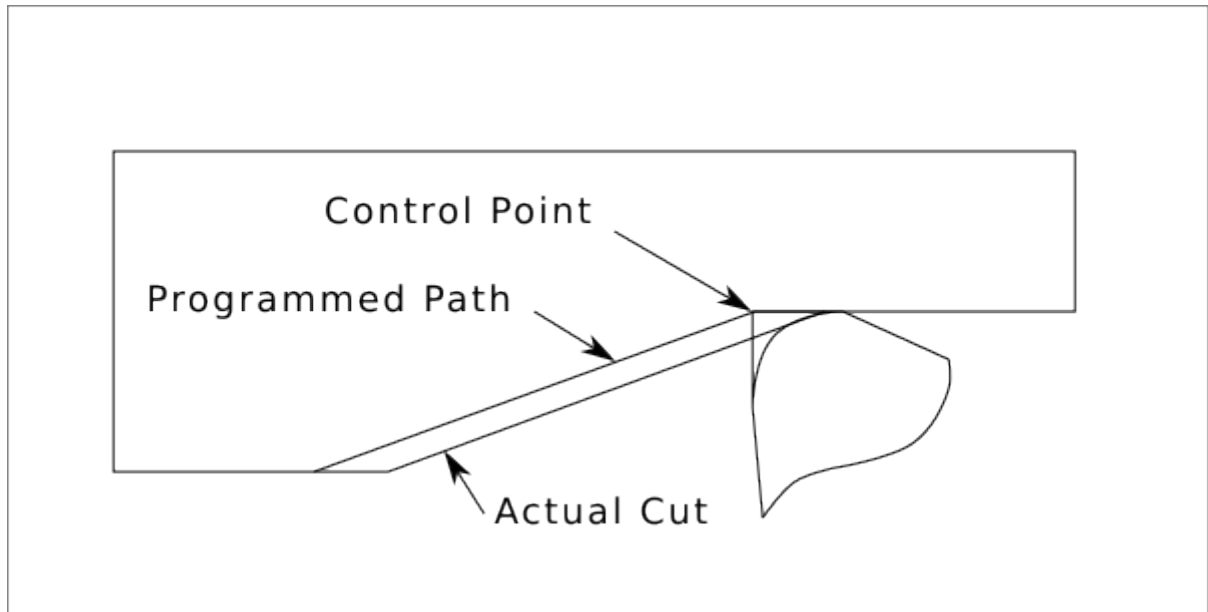


Figure 4.44: Ramp Path

In the above example it is a simple exercise to adjust the programmed path to give the desired actual path by moving the programmed path for the ramp to the left the radius of the tool tip.

4.10.7.3 Cutting a Radius

In this example we will examine what happens during a radius cut without cutter comp. In the next figure you see the tool turning the OD of the part. The control point of the tool is following the programmed path and the tool is touching the OD of the part.

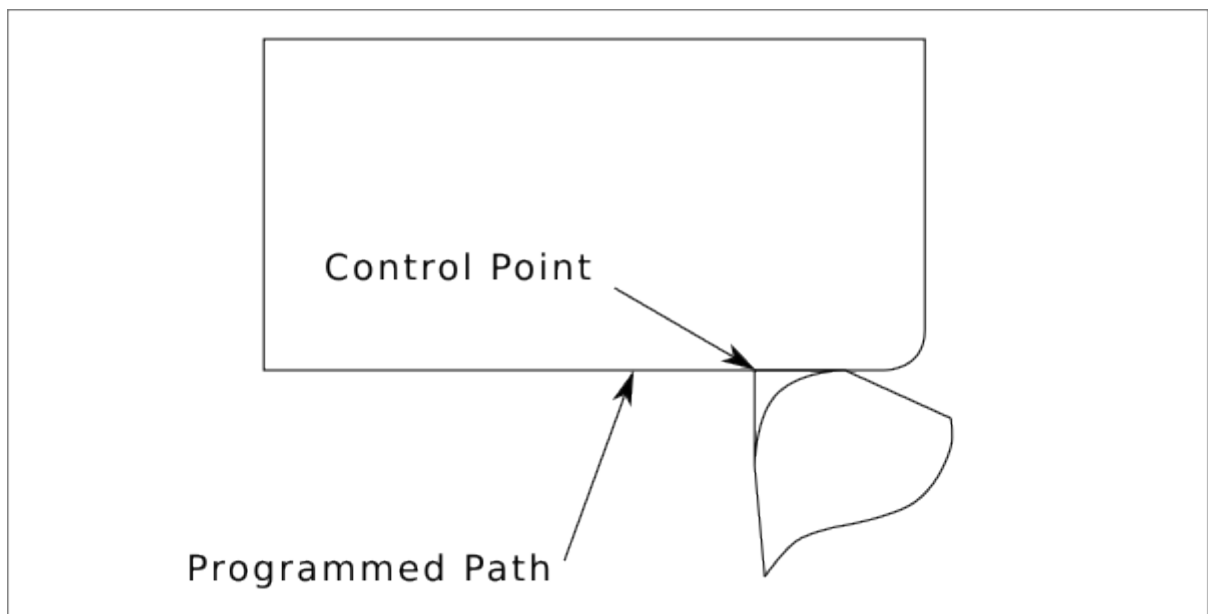


Figure 4.45: Turning Cut

In this next figure you can see as the tool approaches the end of the part the control point still follows the path but the tool tip has left the part and is cutting air. You can also see that even though a radius has been programmed the part will actually end up with a square corner.

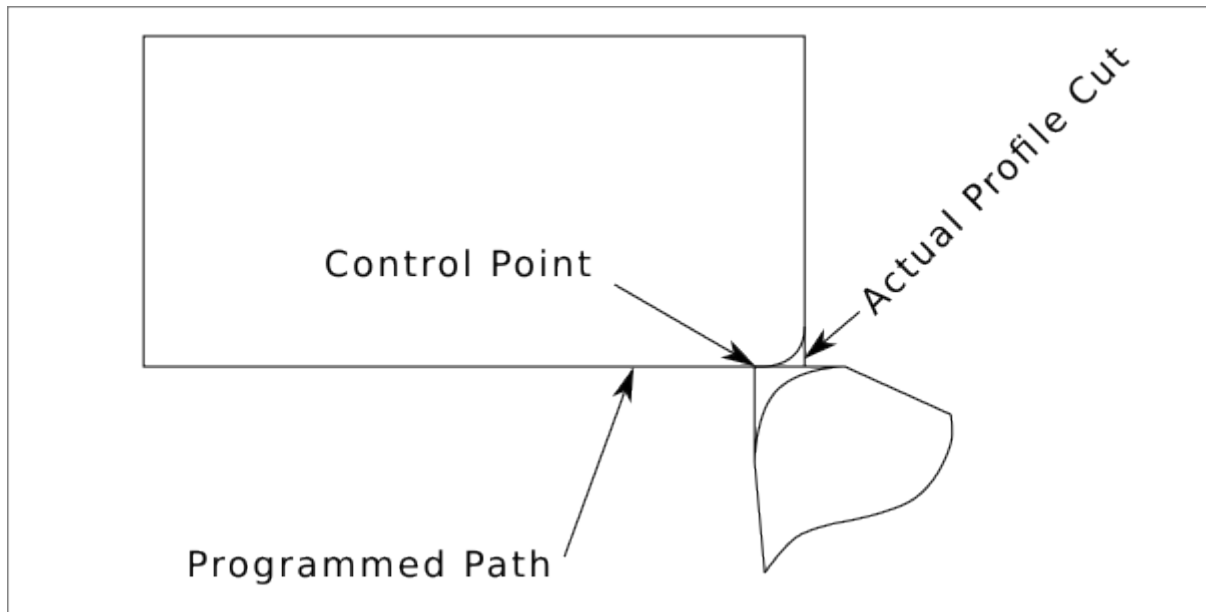


Figure 4.46: Radius Cut

Now you can see as the control point follows the radius programmed the tool tip has left the part and is now cutting air.

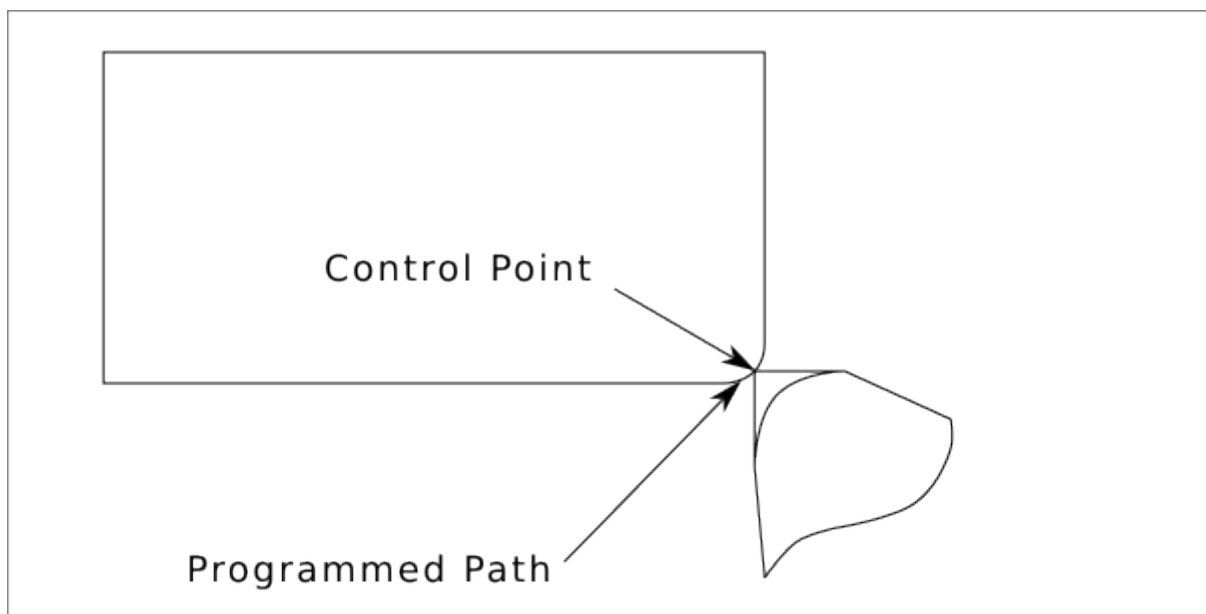


Figure 4.47: Radius Cut

In the final figure we can see the tool tip will finish cutting the face but leave a square corner instead of a nice radius. Notice also that if you program the cut to end at the center of the part a small amount

of material will be left from the radius of the tool. To finish a face cut to the center of a part you have to program the tool to go past center at least the nose radius of the tool.

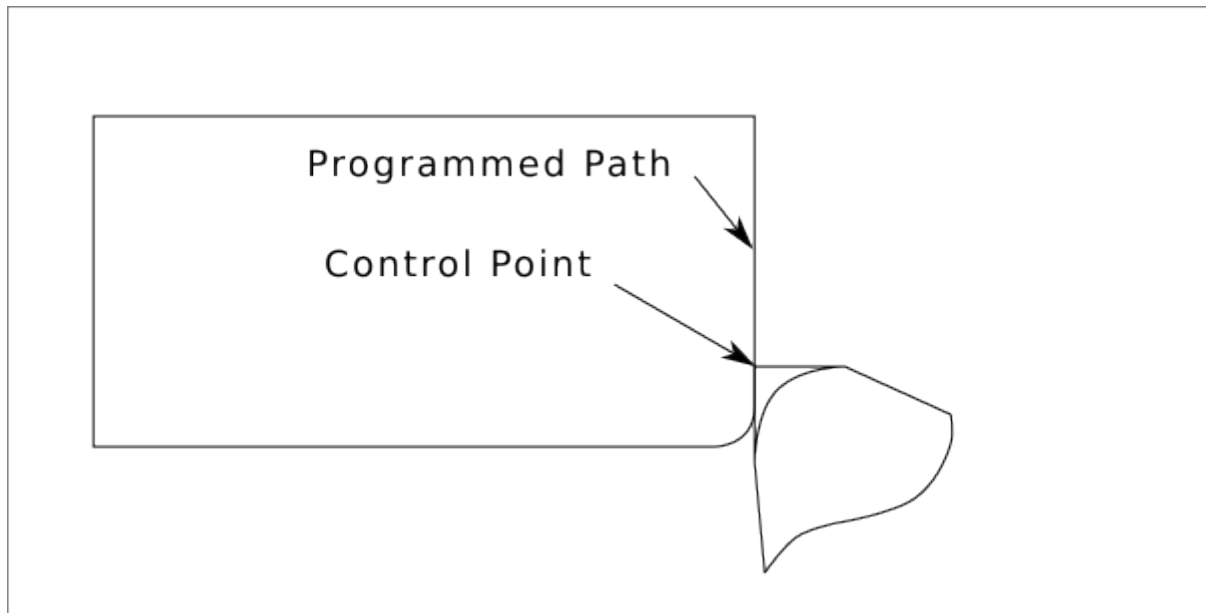


Figure 4.48: Face Cut

4.10.7.4 Using Cutter Comp

- When using cutter comp on a lathe think of the tool tip radius as the radius of a round cutter.
- When using cutter comp the path must be large enough for a round tool that will not gouge into the next line.
- When cutting straight lines on the lathe you might not want to use cutter comp. For example boring a hole with a tight fitting boring bar you may not have enough room to do the exit move.
- The entry move into a cutter comp arc is important to get the correct results.

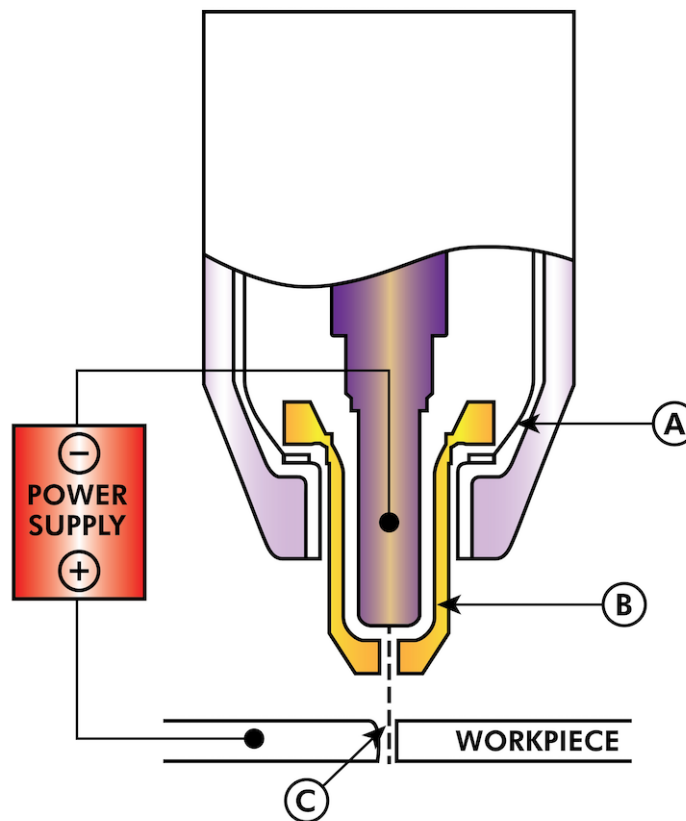
4.11 Plasma Cutting Primer for LinuxCNC Users

4.11.1 What Is Plasma?

Plasma is a fourth state of matter, an ionised gas which has been heated to an extremely high temperature and ionised so that it becomes electrically conductive. The plasma arc cutting and gouging processes use this plasma to transfer an electrical arc to the workpiece. The metal to be cut or removed is melted by the heat of the arc and then blown away. While the goal of plasma arc cutting is the separation of the material, plasma arc gouging is used to remove metals to a controlled depth and width.

Plasma torches are similar in design to the automotive spark plug. They consist of negative and positive sections separated by a center insulator. Inside the torch, the pilot arc starts in the gap between the negatively charged electrode and the positively charged tip. Once the pilot arc has ionised the plasma gas, the superheated column of gas flows through the small orifice in the torch tip, which is focused on the metal to be cut.

In a Plasma Cutting Torch a cool gas enters Zone B, where a pilot arc between the electrode and the torch tip heats and ionises the gas. The main cutting arc then transfers to the workpiece through the column of plasma gas in Zone C. By forcing the plasma gas and electric arc through a small orifice, the torch delivers a high concentration of heat to a small area. The stiff, constricted plasma arc is shown in Zone C. Direct current (DC) straight polarity is used for plasma cutting, as shown in the illustration. Zone A channels a secondary gas that cools the torch. This gas also assists the high velocity plasma gas in blowing the molten metal out of the cut allowing for a fast, slag - free cut.



TYPICAL TORCH HEAD DETAIL

4.11.2 Arc Initialisation

There are two main methods for arc initialisation for plasma cutters that are designed for CNC operation. Whilst other methods are used on some machines (such as scratch start where physical contact with the material is required), they are unsuited for CNC applications..

4.11.2.1 High Frequency Start

This start type is widely employed, and has been around the longest. Although it is older technology, it works well, and starts quickly. But, because of the high frequency high voltage power that is required generated to ionise the air, it has some drawbacks. It often interferes with surrounding electronic circuitry, and can even damage components. Also a special circuit is needed to create a Pilot arc. Inexpensive models will not have a pilot arc, and require touching the consumable to the work to start. Employing a HF circuit also can increase maintenance issues, as there are usually adjustable points that must be cleaned and readjusted from time to time.

4.11.2.2 Blowback Start

This start type uses air pressure supplied to the cutter to force a small piston or cartridge inside the torch head back to create a small spark between the inside surface of the consumable, ionising the air, and creating a small plasma flame. This also creates a "pilot arc" that provides a plasma flame that stays on, whether in contact with the metal or not. This is a very good start type that is now used by several manufacturers. It's advantage is that it requires somewhat less circuitry, is a fairly reliable and generates far less electrical noise.

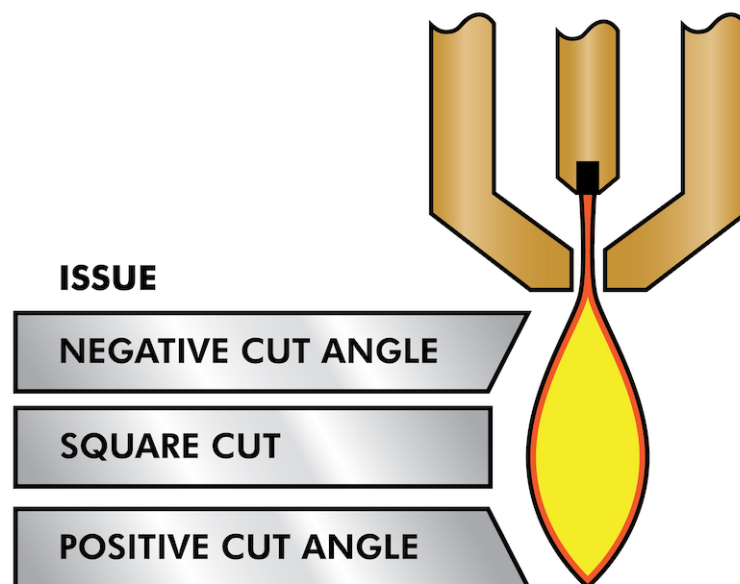
For entry level air plasma CNC systems, the blowback style is much preferred to minimise electrical interference with electronics and standard PCs but the High frequency start still rules supreme in larger machines from 200 amps and up. These require industrial level PC's and electronics and even commercial manufacturers have had issues with faults because they have failed to account for electrical noise in their designs.

4.11.3 CNC Plasma

Plasma operations on CNC machines is quite unique in comparison to milling or turning and is a bit of an orphan process. Uneven heating of the material from the plasma arc will cause the sheet to bend and buckle. Most sheets of metal do not come out of the mill or press in a very even or flat state. Thick sheets (30mm plus) can be out of plane as much as 50mm to 100mm. Most other CNC G-Code operations will start from a known reference or a piece of stock that has a known size and shape and the G-Code is written to rough the excess off and then finally cut the finished part. With plasma the unknown state of the sheet makes it impossible to generate G-Code that will cater for these variances in the material.

A plasma Arc is oval in shape and the cutting height needs to be controlled to minimise bevelled edges. If the torch is too high or too low then the edges can become excessively bevelled. It is also critical that the torch is held perpendicular to the surface.

- **Torch to work distance can impact edge bevel**

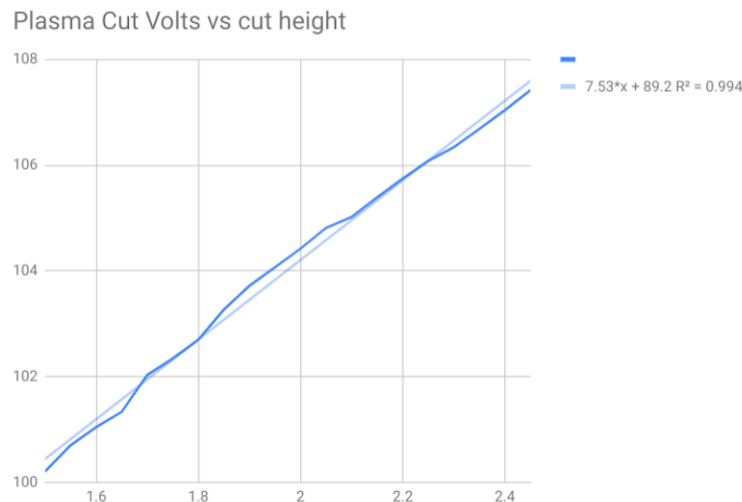


- **Negative cut angle:** torch too low, increase torch to work distance.
- **Positive cut angle:** torch too high, decrease torch to work distance.

Note

A slight variation in cut angles may be normal, as long as it is within tolerance.

The ability to precisely control the cutting height in such a hostile and ever changing environment is a very difficult challenge. Fortunately there is a very linear relationship between Torch height (Arc length) and arc voltage as this graph shows.



This graph was prepared from a sample of about 16,000 readings at varying cut height and the regression analysis shows 7.53 volts per mm with 99.4% confidence. In this particular instance this sample was taken from an Everlast 50 amp machine being controlled by Linuxcnc.

Torch voltage then becomes an ideal process control variable to use to adjust the cut height. Let's assume for simplicity that voltage changes by 10 volts per mm. This can be restated to be 1 volt per 0.1mm (0.004"). Major plasma machine manufacturers (eg Hypertherm, Thermal Dynamics and ESAB), produce cut charts that specify the recommended cut height and estimated arc voltage at this height as well as some additional data. So if the arc voltage is 1 volt higher than the manufacturers specification, the controller simply needs to lower the torch by 0.1 mm (0.004") to move back to the desired cut height. A torch height control unit (THC) is traditionally used to manage this process.

4.11.4 Choosing a Plasma Machine for CNC operations

There are a plethora of plasma machines available on the market today and not all of them are suited for CNC use. CNC Plasma cutting is a complex operation and it is recommended that integrators choose a suitable plasma machine. Failure to do this is likely to cause hours and hours of fruitless trouble shooting trying to work around the lack of what many would consider to be mandatory features.

Whilst rules are made to be broken if you fully understand the reasons the rule apply, we consider a new plasma table builder should select a machine with the following features:

- Blowback start to minimise electrical noise to simplify construction
- A Machine torch is preferred but many have used hand torches.
- A fully shielded torch tip to allow ohmic sensing

If you have the budget, a higher end machines will supply:

- Manufacturer provided cut charts which will save many hours and material waste calibrating cut parameters
-

- Dry Contacts for ArcOK
- Terminals for Arc On switch
- Raw arc voltage or divided arc voltage output
- Optionally a RS485 interface if using a Hypertherm plasma cutter and want to control it from the Linuxcnc console.
- Higher duty cycles

In recent times, another class of machine which includes some of these features has become available at around USD \$550. One example is the Herocut55i available on Amazon but there is yet no feedback from users. This Machine features a blowback torch, ArcOK output, torch start contacts and raw arc voltage.

4.11.5 Types Of Torch Height Control

Most THC units are external devices and many have a fairly crude “bit bang” adjustment method. They provide two signals back to the LinuxCNC controller. One turns on if the Z axis should move up and the other turns on if the Z axis should move down. Neither signal is true if the torch is at the correct height. The popular Proma 150 THC is one example of this type of THC. The Linuxcnc THCUD component is designed to work with this type of THC.

With the release of the Mesa THCAD voltage to frequency interface, LinuxCNC was able to decode the actual torch voltage via an encoder input. This allowed LinuxCNC to control the Z axis and eliminate external hardware. Early implementations utilising the THCAD replicated the “bit bang” approach. The Linuxcnc THC component is an example of this approach.

Jim Colt of Hypertherm is on record saying that the best THC controllers were fully integrated into the CNC controller itself. Of course he was referring to high end systems manufactured by Hypertherm, Esab, Thermal Dynamics and others such as Advanced Robotic Technology in Australia, little dreaming that open source could produce systems using this approach that rival high end systems.

The inclusion of external offsets in Linuxcnc V2.8 allowed plasma control in LinuxCNC to rise to a whole new level. External Offsets refers to the ability to apply an offset to the axis commanded position external to the motion controller. This is perfect for plasma THC control as a method to adjust the torch height in real time based on our chosen process control methodology. Following a number of experimental builds, the Plasmac configuration was incorporated into LinuxCNC 2.8. [QtPlasmaC](#) has superceded Plasmac in LinuxCNC 2.9. This has been an extremely ambitious project and many people around the globe have been involved in testing and improving the feature set. QtPlasmaC is unique in that its design goal was to support all THCs including the simple bit bang ones through to sophisticated torch voltage control if the voltage is made available to LinuxCNC via a THCAD or some other voltage sensor. What’s more, QtPlasmaC is designed to be a stand alone system that does not need any additional G-Code subroutines and allows the user to define their own cut charts that are stored in the system and accessible by a drop-down.

4.11.6 Arc OK Signal

Plasma machines that have a CNC interface contain a set of dry contacts (eg a relay) that close when a valid arc is established and each side of these contacts are brought out onto pins on the CNC interface. A plasma table builder should connect one side of these pins to field power and the other to an input pin. This then allows the CNC controller to know when a valid arc is established and also when an arc is lost unexpectedly. There is a potential trap here when the input is a high impedance circuit such as a Mesa card. If the dry contacts are a simple relay, there is a high probability that the current passing through the relay is less than the minimum current specification. Under these conditions, the relay contacts can suffer from a buildup of oxide which over time can result in intermittent contact

operation. To prevent this from happening, a pull down resistor should be installed on the controller input pin. Care should be taken to ensure that this resistor is selected to ensure the minimum current passes through the relay and is of sufficient wattage to handle the power in the circuit. Finally, the resistor should be mounted in such a way that the generated heat does not damage anything whilst in operation.

If you have an ArcOK signal, it is recommended it is used over and above any synthesised signal to eliminate potential build issues. A synthesised signal available from an external THC or QtPlasmaC's Mode 0 can't fully replace the ArcOK circuitry in a plasma inverter. Some build issues have been observed where misconfiguration or incompatibility with the plasma inverter has occurred from a synthesised ArcOK signal. By and large however, a correctly configured synthesised ArcOK signal is fine.

A simple and effective arcOK signal can be achieved with a simple reed relay. Wrap 3 turns of one of the plasma cutter's thick cables (eg the material clamp cable) around it. Place the relay in an old pen tube for protection and connect one side of the relay to field power and the other end to your ArcOK input pin.

4.11.7 Initial Height Sensing

Because the cutting height is such a critical system parameter and the material surface is inherently uneven, a Z axis mechanism needs a method to sense the material surface. There are three methods this can be achieved; Current sensing to detect increased motor torque, a "float" switch and an electrical or "ohmic" sensing circuit that is closed when the torch shield contacts the material. Current sensing is not a viable technique for DIY tables but float switches and ohmic sensing are discussed below:

4.11.7.1 Float Switches

The torch is mounted on a sliding stage that can move up when the torch tip contacts the material surface and trigger a switch or sensor. Often this is achieved under G-Code control using the G38 Commands. If this is the case, then after initial probing, it is recommended to probe away from the surface until the probe signal is lost at a slower speed. Also, ensure the switch hysteresis is accounted for.

Regardless of the probing method used, it is strongly recommended that float switch is implemented so that there is a fallback or secondary signal to avoid damage to the torch from a crash.

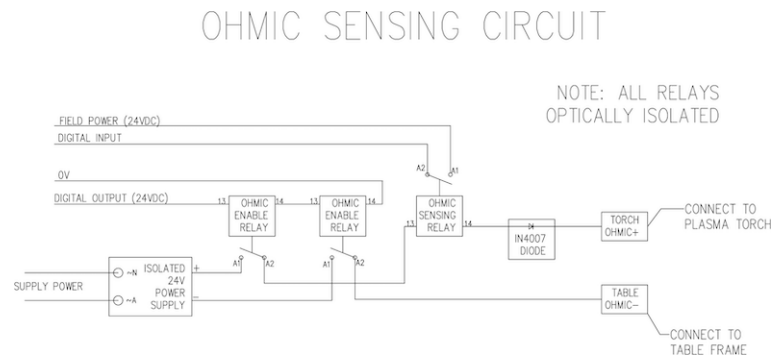
4.11.7.2 Ohmic Sensing

Ohmic sensing relies on contact between the torch and the material acting as a switch to activate an electrical signal that is sensed by the CNC controller. Provided the material is clean, this can be a much more accurate method of sensing the material than a float switch which can cause deflection of the material surface. This ohmic sensing circuit is operating in an extremely hostile environment so a number of failsafes need to be implemented to ensure safety of both the CNC electronics and the operator. In plasma cutting, the earth clamp attached to the material is positive and the torch is negative. It is recommended that:

1. Ohmic sensing only be implemented where the torch has a shield that is isolated from the torch tip that conveys the cutting arc.
 2. The ohmic circuit uses a totally separate isolated power supply that activates an opto-isolated relay to enable the probing signal to be transmitted to the CNC controller.
 3. The positive side of the circuit should be at the torch
-

4. Both sides of the circuit needs to be isolated by opto-isolated relays until probing is being undertaken
5. Blocking diodes be used to prevent arc voltage entering the ohmic sensing circuit.

The following is an example circuit that has been proven to work and is compatible with the Linuxcnc QtPlasmaC configuration.



4.11.7.3 Hypersensing with a MESA THCAD-5

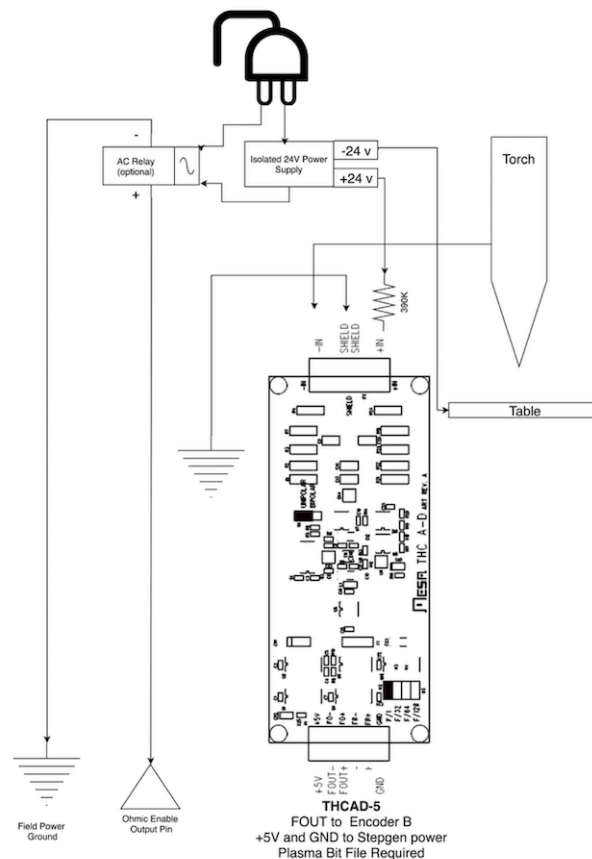
A more sophisticated method of material sensing that eliminates the relays and diodes is to use another THCAD-5 to monitor the material sensing circuit voltage from an isolated power supply. The advantage this has is the THCAD is designed for the hostile plasma electrical environment and totally and safely isolates the logic side from the high voltage side.

To implement this method, a second encoder input is required.

If using a mesa card, different firmware is available to provide 2 additional Encoder A inputs on the Encoder B and Encoder Index pins. This firmware is available for download for the 7i76e and 7i96 boards from the Mesa web site on the product pages.

The THCAD is sensitive enough to see the ramp up in circuit voltage as contact pressure increases. The ohmic.comp component included in Linuxcnc can monitor the sensing voltage and set a voltage threshold above which it is deemed contact is made and an output is enabled. By monitoring the voltage, a lower “break circuit” threshold can be set to build in strong switch hysteresis. This minimises false triggering. In our testing, we found the material sensing using this method was more sensitive and robust as well as being simpler to implement the wiring. One further advantage is using software outputs instead of physical I/O pins is that it frees up pins to use for other purposes. This advantage is helpful to get the most out of the Mesa 7i96 which has limited I/O pins.

The following circuit diagram shows how to implement a hypersensing circuit.



We used a 15W Mean Well HDR-15 Ultra Slim DIN Rail Supply 24V DIN rail based isolated power supply.. This is a double insulated Isolation Class II device that will withstand any arc voltage that might be applied to the terminals.

4.11.7.4 Example HAL Code for Hypersensing

The following HAL code can be pasted into your QtPlasmaC's custom.hal to enable Ohmic sensing on Encoder 2 of a 7i76e. Install the correct bit file and connect the THCAD to IDX+ and IDX-. Be sure to change the calibration settings to agree with your THCAD-5.

```
# --- Load the Component ---
loadrt ohmic names=ohmicsense
addf ohmicsense servo-thread

# --- 7i76e ENCODER 2 SETUP FOR OHMIC SENSING---
setp hm2_7i76e.0.encoder.02.scale -1
setp hm2_7i76e.0.encoder.02.counter-mode 1

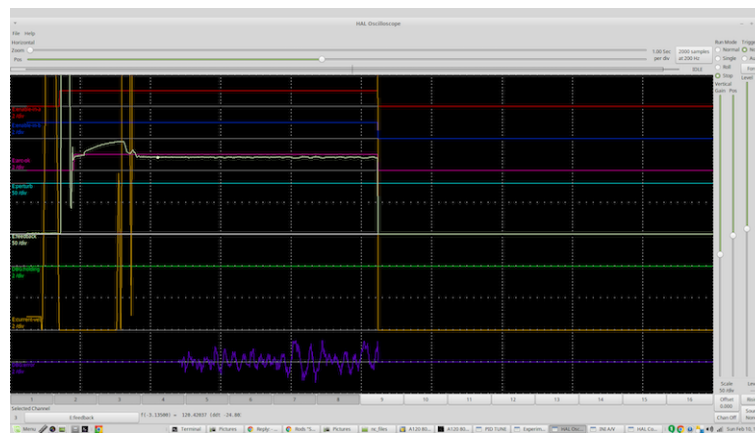
# --- Configure the component ---
setp ohmicsense.thcad-0-volt-freq 140200
setp ohmicsense.thcad-max-volt-freq 988300
setp ohmicsense.thcad-divide 32
setp ohmicsense.thcad-fullscale 5
```

```
setp ohmicsense.volt-divider      4.9
setp ohmicsense.ohmic-threshold  22.0
setp ohmicsense.ohmic-low        1.0
net ohmic-vel ohmicsense.velocity-in <= hm2_7i76e.0.encoder.02.velocity

# --- Replace QtPlasmaC's Ohmic sensing signal ---
unlinkp debounce.0.2.in
net ohmic-true ohmicsense.ohmic-on => debounce.0.2.in
net plasmac:ohmic-enable    => ohmicsense.is-probing
```

4.11.8 THC Delay

When an arc is established, arc voltage peaks significantly and then settles back to a stable voltage at cut height. As shown by the green line in the image below.



It is important for the plasma controller to “wait it out” before auto sampling the torch voltage and commencing THC control. If enabled too early, the voltage will be above the desired cut volts and the torch will be driven down in an attempt to address a perceived over-height condition.

In our testing this varies between machines and material from 0.5 to 1.5 seconds. Therefore a delay of 1.5 seconds after a valid arcOK signal is received before enabling THC control is a safe initial setting. If you want to shorten this for a given material, LinuxCNC’s Halscope will allow you to plot the torch voltage and make informed decisions about the shortest safe delay is used.

Note

If the cut velocity is not near the desired cut speed at the end of this delay, the controller should wait until this is achieved before enabling the THC.

4.11.9 Torch Voltage Sampling

Rather than relying on the manufacturer’s cut charts to set the desired torch voltage, many people (the writer included) prefer to sample the voltage as the THC is enabled and use that as a set point.

4.11.10 Torch Breakaway

It is recommended that a mechanism is provided to allow the torch to “break away” or fall off in the case of impact with the material or a cut part that has tipped up. A sensor should be installed to allow the CNC controller to detect if this has occurred and pause the running program. Usually a break away is implemented using magnets to secure the torch to the Z axis stage.

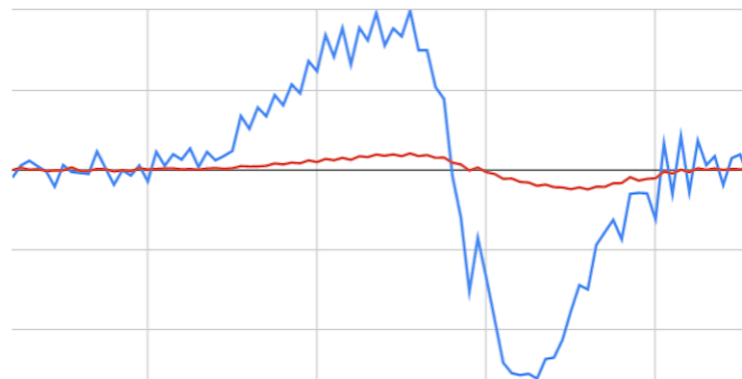
4.11.11 Corner Lock / Velocity Anti-Dive

The Linuxcnc trajectory planner is responsible for translating velocity and acceleration commands into motion that obeys the laws of physics. For example, motion will slow when negotiating a corner. Whilst this is not a problem with milling machines or routers, this poses a particular problem for plasma cutting as the arc voltage increases as motion slows. This will cause the THC to drive the torch down. One of the enormous advantages of a THC control embedded within the LinuxCNC motion controller is that it knows what is going on at all times. So it becomes a trivial matter to monitor the current velocity (`motion.current-velocity`) and to hold THC operation if it falls below a set threshold (eg, 10% below the desired feedrate).

4.11.12 Void / Kerf Crossing

If the plasma torch passes over a void while cutting, arc voltage rapidly rises and the THC responds by violent downward motion which can smash the torch into the material possibly damaging it. This is a situation that is difficult to detect and handle. To a certain extent it can be mitigated by good nesting techniques but can still occur on thicker material when a slug falls away. This is the one problem that has yet to be solved within the LinuxCNC open source movement.

One suggested technique is to monitor the rate of change in torch volts over time (dv/dt) because this parameter is orders of magnitude higher when crossing a void than what occurs due to normal warpage of the material. The following graph shows a low resolution plot of dv/dt (in blue) while crossing a void. The red curve is a moving average of torch volts.



So it should be possible to compare the moving average with the dv/dt and halt THC operation once the dv/dt exceeds the normal range expected due to warpage. More work needs to be done in this area to come up with a working solution in LinuxCNC.

4.11.13 Hole And Small Shape Cutting

It is recommended that you slow down cutting when cutting holes and small shapes.

John Moore says: "If you want details on cutting accurate small holes look up the sales sheets on Hypertherm's "True Hole Technology" also look on PlasmaSpider, user seanp has posted extensively on his work using simple air plasma.

The generally accepted method to get good holes from 37mm dia. and down to material thickness with minimal taper using an air plasma is:

1. Use recommended cutting current for consumables.
2. Use fixed (no THC) recommended cutting height for consumables.

3. Cut from 60% to 70% of the recommended feed rate of consumables and materials.
4. Start lead in at or near center of hole.
5. Use perpendicular lead in.
6. No lead out, either a slight over burn or early torch off depending on what works best for you.

You will need to experiment to get exact hole size because the kerf with this method will be wider than your usual straight cut.”

This slow down can be achieved by manipulating the feed rate directly in your post processor or by using adaptive feed and an analog pin as input. This lets you use M67/M68 to set the percentage of desired feed to cut at.

- **Knowing The Feedrate**

From the preceding discussion it is evident that the plasma controller needs to know the feed rate set by the user. This poses a problem with LinuxCNC because the Feedrate is not saved by LinuxCNC after the G-Code is buffered and parsed. There are two approaches to work around this:

1. Remap the F command and save the commanded feedrate set in G-Code via an M67/M68 command
2. Storing the cut charts in the plasma controller and allow the current feedrate be queried by the G-Code program (as QtPlasmaC does)

One experimental Linuxcnc branch that would be useful for plasma cutting was the state tags branch. This adds a “tag” that is available to motion containing the current feed and speed rates for all active motion commands. It has been merged and will be in LinuxCNC v2.9

4.11.14 I/O Pins For Plasma Controllers

Plasma cutters require several additional pins. In LinuxCNC, there are no hard and fast rules about which pin does what. In this discussion we will assume the plasma inverter has a CNC interface and the controller card has active high inputs are in use (Eg. Mesa 7i76e).

Plasma tables can be large machines and we recommend that you take the time to install separate max/min limit switches and homing switches for each joint. The exception might be the Z axis lower limit. When a homing switch is triggered the joint decelerates fairly slowly for maximum accuracy. This means that if you wish to use homing velocities that are commensurate with table size, you can overshoot the initial trigger point by 50-100mm. If you use a shared home/limit switch, you have to move the sensor off the trigger point with the final HOME_OFFSET or you will trigger a limit switch fault as the machine comes out of homing. This means you could lose 50mm or more of axis travel with shared home/limit switches. This does not happen if separate home and limit switches are used.

The following pins are usually required (note that suggested connections may not be appropriate for a QtPlasmaC configuration):

4.11.14.1 Arc OK (input)

- Inverter closes dry contacts when a valid arc is established
 - Connect Field power to one Inverter ArcOK terminal.
 - Connect other Inverter Ok Terminal to input pin.
 - Usually connected to one of the *motion.digital-<nn>* pins for use from G-Code with M66
-

4.11.14.2 Torch On (output)

- Triggers a relay to close the torch on switch in the inverter
- Connect the torch on terminals on the inverter to the relay output terminals
- Connect one side of the coil to the output pin
- Connect the other side of the coil to Field Power ground.
- If a mechanical relay is used, connect a flyback diode (eg IN400x series) across the coil terminals with the band on the diode pointing towards the output pin
- If a Solid State Relay is used, polarity may need to be observed on the outputs
- In some circumstances, the onboard spindle relay on a Mesa card can be used instead of an external relay.
- Usually connected to *spindle.0.on*



Warning

It is strongly recommended that the torch cannot be enabled while this pin is false otherwise the torch will not be extinguished when estop is pressed;

4.11.14.3 Float switch (input)

- Used for surface probing. A sensor or switch that is activated if the torch slides up when it hits the material.
- Connect Proximity sensor output to chosen input pin. If mechanical switches are used. Connect one side of the switch to field power and the other side of the switch to input.
- Usually Connected to *motion.probe-input*

4.11.14.4 Ohmic Sensor enable (output)

- See the [ohmic sensing](#) schematic.
- Connect output pin to one side of the isolation relays and the other side to field power ground.
- In a non-QtPlasmaC configuration, usually triggered by a *motion.digital-out-<nn>* so it can be controlled in G-Code by M62/M63/M64/M65

4.11.14.5 Ohmic Sensing (input)

- Take care to follow the [ohmic sensing](#) schematic shown previously.
 - An isolated power supply triggers a relay when the torch shield contacts the material.
 - Connect field power to one output terminal and the other to the input.
 - Take care to observe relay polarity if opto-coupled solid State relays are used.
 - Usually connected to *motion.probe-input* and may be or'd with the float switch.
-

As can be seen, plasma tables are pin intensive and we have already consumed about 15 inputs before the normal estops are added. Others have other views but it is the writer's opinion that the Mesa 7i76e is preferred over the cheaper 7i96 to allow for MPG's, scale and axis selection switch and other features you may wish to add over time. If your table uses servos, there are a number of alternatives. Whilst there are other suppliers, designing your machine around the Mesa ecosystem will simplify use of their THCAD board to read arc voltage.

4.11.14.6 Torch Breakaway Sensor

- As mentioned earlier, a breakaway sensor should be installed that is triggered if the torch crashes and falls off.
- Usually, this would be connected to *halui.program-pause* so the fault can be rectified and the program resumed.

4.11.15 G-Code For Plasma Controllers

Most plasma controllers offer a method to change settings from G-Code. Linuxcnc support this via M67/M68 for analog commands and M62-M65 for digital (on/off commands). How this is implemented is totally arbitrary. Lets look at how the LinuxCNC QtPlasmaC configuration does this:

4.11.15.1 Select Material Settings In QtPlasmaC And Use The Feedrate For That Material:

```
M190 Pn
M66 P3 L3 Q1
F#<_hal[plasmac.cut-feed-rate]>
M3 S1
```

Note

Users with a very large number of entries in the QtPlasmaC Materials Table may need to increase the Q1 parameter (eg Q2)

4.11.15.2 Enable/Disable THC Operation:

```
M62 P2 will disable THC (synchronised with motion)
M63 P2 will enable THC(synchronised with motion)
M64 P2 will disable THC (immediately)
M65 P2 will enable THC(immediately)
```

4.11.15.3 Reduce Cutting Speeds: (eg for hole cutting)

```
M67 E3 Q0 would set the velocity to 100% of requested~speed
M67 E3 Q40 would set the velocity to 40% of requested~speed
M67 E3 Q60 would set the velocity to 60% of requested~speed
M67 E3 Q100 would set the velocity to 100% of requested~speed
```

4.11.15.4 Cutter Compensation:

```
G41.1 D#<_hal[plasmac_run.kerf-width-f]> ; for left of programmed path
G42.1 D#<_hal[plasmac_run.kerf-width-f]> for right of programmed path
G40 to turn compensation off
```

Note

Integrators should familiarise themselves with the Linuxcnc documentation for the various Linuxcnc G-Code commands mentioned above.

4.11.16 External Offsets and Plasma Cutting

External Offsets were introduced to Linuxcnc with version 2.8. By external, it means that we can apply an offset external to the G-Code that the trajectory planner knows nothing about. It easiest to explain with an example. Picture a lathe with an external offset being applied by a mathematical formula to machine a lobe on a cam. So the lathe is blindly spinning around with the cut diameter set to a fixed diameter and the external offset moves the tool in and out to machine the cam lobe via an applied external offset. To configure our lathe to machine this cam, we need to allocate some portion of the axis velocity and acceleration to external offsets or the tool can't move. This is where the ini variable `OFFSET_AV_RATIO` comes in. Say we decide we need to allocate 20% of the velocity and acceleration to the external offset to the Z axis. We set this equal to 0.2. The consequence of this is that your maximum velocity and acceleration for the Lathe's Z axis is only 80% of what it could be.

External offsets are a very powerful method to make torch height adjustments to the Z axis via a THC. But plasma is all about high velocities and rapid acceleration so it makes no sense to limit these parameters. Fortunately in a plasma machine, the Z axis is either 100% controlled by the THC or it isn't. During the development of Linuxcnc's external offsets it was recognised that Z axis motion by G-Code and by THC were mutually exclusive. This allows us to trick external offsets into giving 100 % of velocity and acceleration all of the time. We can do this by doubling the machine's Z axis velocity and acceleration settings in the ini file and set `OFFSET_AV_RATIO = 0.5`. That way 100% of the maximum velocity and acceleration will be available for both probing and THC.

Example: On a metric machine with a NEMA23 motor with a direct drive to a 5mm ball screw, 60 mm/second maximum velocity and 700 mm/sec/sec acceleration were determined to be safe values without loss of steps. For this machine, set the Z axis in the ini file as follows:

```
[AXIS_Z]
OFFSET_AV_RATIO = 0.5
MAX_VELOCITY = 120
MAX_ACCELERATION = 1400
```

The joint associated with this axis would have the velocity and acceleration variables set as follows:

```
[JOINT_n]
MAX_VELOCITY = 60
MAX_ACCELERATION = 700
```

For further information about external offsets (for V 2.8 and higher) please read the [\[AXIS_<letter>\] Section](#) of the INI file document and [External Axis Offsets](#) in the Linuxcnc documentation.

4.11.17 Reading Arc Voltage With The Mesa THCAD

The Mesa THCAD board is a remarkably well priced and accurate voltage to frequency converter that is designed for the hostile noisy electrical environment associated with plasma cutting. Internally it

has a 0-10 volt range. This range can be simply extended by the addition of some resistors as described in the documentation. This board is available in three versions, the newer THCAD-5 with a 0-5 volt range, the THCAD-10 with a 0-10 volt range and the THCAD-300 which is pre-calibrated for a 300 volt extended range. Each board is individually calibrated and a sticker is applied to the board that states the frequency at 0 volts and full scale. For use with LinuxCNC, it is recommended that the 1/32 divisor be selected by the appropriate link on the board. In this case, be sure to also divide the stated frequencies by 32. This is more appropriate for the 1 kHz servo thread and also allows more time for the THCAD to average and smooth the output.

There is a lot of confusion around how to decode the THCAD output so lets consider the Mesa 7i76e and the THCAD-10 for a moment with the following hypothetical calibration data:

- Full scale 928,000 Hz (1/32 29,000 Hz)
- 0 volt 121,600 Hz (1/32 3,800 Hz)

Because the full scale is 10 volts, then the frequency per volt is:

$$(29,000 - 3,800) / 10 = 2,520 \text{ Hz per volt}$$

So assuming we have a 5 volt input, the calculated frequency would be:

$$(2520 * 5) + 3,800 = 16,400$$

So now it should be fairly clear how to convert the frequency to its voltage equivalent:

$$\text{Volts} = (\text{frequency} - 3,800) / 2,520$$

4.11.17.1 THCAD Connections

On the high voltage side:

- Connect the divided or raw arc voltage to IN + and IN-
- Connect the interconnect cable shield to the Shield connection.
- Connect the other Shield terminal to frame ground.

Assuming it is connected to a 7i76e, connect the output to the spindle encoder input:

- THCAD +5v to TB3 Pin 6 (+5 VP)
- THCAD -5v to TB3 Pin 1 (GND)
- THCAD FOUT+ to TB3 Pin 7 (ENC A+)
- THCAD FOUT- to TB3 Pin 8 (ENC A-)

4.11.17.2 THCAD Initial Testing

Make sure you have the following lines in your ini file (assuming a Mesa 7i76e):

```
setp hm2_7i76e.0.encoder.00.scale -1
setp hm2_7i76e.0.encoder.00.counter-mode 1
```

Power up your controller and open Halshow (Axis: Show Homing Configuration), Drill down to find the *hm2_7i76e.0.encoder.00.velocity pin*. With 0 volts applied, it should be hovering around the 0 volt frequency (3,800 in our example). Grab a 9 volt battery and connect it to IN+ and IN-. For a THCAD-10 you can now calculate the expected velocity. (26,480 in our hypothetical example). If you pass this test, then you are ready to configure your LinuxCNC plasma controller.

4.11.17.3 Which Model THCAD To Use

The THCAD-5 is useful if you intend to use it for ohmic sensing. There is no doubt the THCAD-10 is the more flexible device and it is easy to alter the scaling. However, there is one caveat that can come into play with some cheaper plasma cutters with an inbuilt voltage divider. That is, the internal resistors may be sensed by the THCAD as being part of its own external resistance and return erroneous results. For example, the 16:1 divider on the Everlast plasma cutters needs to be treated as 24:1 (and 50:1 becomes 75:1). This is not a problem with more reputable brands (eg. Thermal Dynamics, Hypertherm, ESAB etc). So if you are seeing lower than expected cutting voltages, it might be preferable to reconfigure the THCAD to read raw arc voltage.

Remembering that plasma arc voltages are potentially lethal, here are some suggested criteria.

Pilot Arc Start Because there is not likely to be any significant EMI, you should be able to safely install the THCAD in your control panel if you have followed our construction guidelines.

- If you do not have a voltage divider, either install scaling resistors inside the plasma cutter and install the THCAD in the control panel or follow the suggestions for HF start machines.
- If you have a voltage divider, install a THCAD-10 in your control panel. We've had no problems with this configuration with a 120 amp Thermal Dynamics plasma cutter.

HF Start Install the THCAD at the inverter as the frequency signal is far more immune to EMI noise.

- If you do not have a voltage divider and you have room inside the plasma cutter, install a THCAD-300 inside the plasma cutter.
- If you do not have a voltage divider and you do not have room inside the plasma cutter, install a THCAD-10 in a metal case outside the plasma cutter and install 50% of the scaling resistance on each of the IN+ and IN- inside the plasma cutter case so no lethal voltages come out of the case.
- If you have a voltage divider, install a THCAD-10 in a metal case outside the plasma cutter

Raw Arc voltage presented on a connector In this case, regardless of the arc starting method, there are probably already resistors included in the circuitry to avoid lethal shocks so a THCAD-10 is advised so this resistance (typically 200k Ohms) can be accounted for when choosing a scaling resistor as these resistors will distort the voltage reported by the THCAD-300.

4.11.18 Post Processors And Nesting

Plasma is no different to other CNC operations in that it is:

1. Designed in CAD (where it is output as a DXF or sometimes SVG format).
2. Processed in CAM to generate final G-Code that is loaded to the machine
3. Cutting the parts via CNC G-Code commands.

Some people achieve good results with Inkscape and G-Code tools but SheetCam is a very well priced solution and there are a number of post processors available for Linuxcnc. SheetCam has a number of advanced plasma features designed for plasma cutting and for the price, is a no brainer for anybody doing regular plasma cutting.

4.11.19 Designing For Noisy Electrical Environments

Plasma cutting is inherently an extremely hostile and noisy electrical environment. If you have EMI problems things won't work correctly. You might fire the torch and the computer will reboot in a more obvious example, but you can have any number of other odd symptoms. They will pretty much all happen only when the torch is cutting - often when it is first fired.

Therefore, system builders should select components carefully and design from the ground up to cope with this hostile environment to avoid the impact of Electro-Magnetic Interference (EMI). Failure to do this could result in countless hours of fruitless troubleshooting.

Choosing ethernet boards such as the Mesa 7i76e or the cheaper 7i96 helps by allowing the PC to be located away from the electronics and the plasma machine. This hardware also allows the use of 24 volt logic systems which are much more noise tolerant. Components should be mounted in a metal enclosure connected to the mains earth. It is strongly recommended that an EMI filter is installed on the mains power connection. The simplest way is to use a EMI filtered mains power IEC connector commonly used on PC's and electric appliances which allows this to be achieved with no extra work. Plan the layout of components in the enclosure so that mains power, high voltage motor wires and logic signals are kept as separate as possible from each other. If they do have to cross, keep them at 90 degrees.

Peter Wallace from Mesa Electronics suggests; "If you have a CNC compatible plasma source with a voltage divider, I would mount the THCAD inside your electronics enclosure with all the other motion hardware. If you have a manual plasma source and you are reading raw plasma voltage, I would mount the THCAD as close to the plasma source as possible (even inside the plasma source case if it fits.) In this case, make sure that all low side THCAD connections are fully isolated from the plasma source. If you use a shielded box for the THCAD, the shield should connect to your electronic enclosure ground, not the plasma source ground."

It is recommended to run a separate earth wire from motor cases and the torch back to a central star grounding point on the machine. Connect the plasma ground lead to this point and optionally an earth rod driven into the ground as close as possible to the machine (particularly if its a HF start plasma machine).

External wiring to motors should be shielded and appropriately sized to handle the current passing through the circuit. The shield should be left unconnected at the motor end and earthed at the control box end. Consider using an additional pin on any connectors into the control box so the earth can be extended through into the control box and earthed to the chassis right at the stepper/servo motor controller itself.

We are aware of at least one commercial system builder who has had problems with induced electrical noise on the ohmic sensing circuit. Whilst this can be mitigated by using ferrite beads and coiling the cable, adding a feed through power line filter is also recommended where the ohmic sensing signal enters the electronics enclosure.

Tommy Berisha, the master of building plasma machines on a budget says: "If on a budget, consider using old laptop power bricks. They are very good, filtering is good, completely isolated, current limited (this becomes very important when something goes wrong), and fitting 2 or 3 of them in series is easy as they are isolated (be aware that some do have the grounding wired to the negative output terminal, so. It has to be disconnected, simply done by using a power cable with no ground contacts)".

4.11.20 Water Tables

The minimum water level under the cut level of the torch should be around 40mm, having space under slats is nice so the water can level and escape during cutting, having a bit of water above the metal plate being cut is really nice as it gets rid of the little bit of dust, running it submerged is the best way but not preferable for systems with part time use as it will corrode the torch. Adding baking soda to the water will keep the table in a nice condition for many years as it does not allow corrosion while the slats are under water and it also reduces the smell of water vapour. Some people use a water

reservoir with a compressed air inlet so they can push the water from the reservoir up to the water table on demand and thus allow changes in water levels.

4.11.21 Downdraft Tables

Many commercial tables utilise a down draft design so fans are used to suck air down through the slats to capture fumes and sparks. Often tables are zoned so only a section below the torch is opened to the outgoing vent, often using air rams and air solenoids to open shutters. Triggering these zones is relatively straightforward if you use the axis or joint position from one of the motion pins and the lincurve component to map downdraft zones to the correct output pin.

4.11.22 Designing For Speed And Acceleration

In plasma cutting, speed and acceleration are king. The higher the acceleration, the less the machine needs to slow down when negotiating corners. This implies that the gantry should be as light as possible without sacrificing torsional stiffness. A 100mm x 100mm x 2mm aluminium box section has equivalent torsional stiffness to an 80mm x 80mm T slot extrusion yet is 62% lighter. So does the convenience of T slots outweigh the additional construction?

4.11.23 Distance Travelled Per Motor Revolution

Stepper motors suffer from resonance and a direct drive pinion is likely to mean that the motor is operating under unfavourable conditions. Ideally, for plasma machines a distance of around 15-25mm per motor revolution is considered ideal but even around 30mm per revolutions is still acceptable. A 5mm pitch ball screw with a 3:1 or 5:1 reduction drive is ideal for the Z axis.

4.11.24 QtPlasmaC LinuxCNC Plasma Configuration

The [QtPlasmaC](#) which is comprised of a HAL component (plasmac.hal) plus a complete configurations for the QtPlasmaC GUI has received considerable input from many in the LinuxCNC Open Source movement that have advanced the understanding of plasma controllers since about 2015. There has been much testing and development work in getting QtPlasmaC to its current working state. Everything from circuit design to G-Code control and configuration has been included. Additionally, QtPlasmaC supports external THC's such as the Proma 150 but really comes into its own when paired with a Mesa controller as this allows the integrator to include the Mesa THCAD voltage to frequency converter which is purpose built to deal with the hostile plasma environment.

QtPlasmaC is designed to stand alone and includes the ability to include your cutting charts yet also includes features to be used with a post processor like SheetCam.

The QtPlasmaC system is now included in Version 2.9 and above of Linuxcnc. Its now quite mature and has been significantly enhanced since the first version of this guide was written. QtPlasmaC will define LinuxCNC's plasma support for many years to come as it includes all of the features a proprietary high end plasma control system at an open source price.

4.11.25 Hypertherm RS485 Control

Some Hypertherm plasma cutters have a RS485 interface to allow the controller (eg. Linuxcnc) to set amps, pressure and mode. A number of people have used a userspace component written in Python to achieve this. More recently, QtPlasmaC now supports this interface natively. Refer to the QtPlasmaC documentation for how to use it.

The combination of a slow baud rate used by Hypertherm and the userspace component, make this fairly slow to alter machine states so it generally not viable to change settings on the fly while cutting.

When selecting a RS485 interface to use at the PC end, users have reported that USB to RS485 interfaces are not reliable. Good reliable results have been achieved using a hardware based RS232 interface (eg PCI/PCIE or motherboard port) and an appropriate RS485 converter. Some users have reported success with a Sunix P/N: SER5037A PCI RS2322 card a generic XC4136 RS232 to RS485 converter (which may sometimes include a USB cable as well).

4.11.26 Post Processors For Plasma Cutting

CAM programs (Computer Aided Manufacture) are the bridge between CAD (Computer Aided Design) and the final CNC (Computer Numerical Control) operation. They often include a user configurable post processor to define the code that is generated for a specific machine or dialect of G-Code.

Many Linuxcnc users are perfectly happy with using Inkscape to convert .SVG vector based files to G-Code. If you are using a plasma for hobby or home use, consider this option. However, if your needs are more complex, probably the best and most reasonably priced solution is SheetCam. SheetCam supports both Windows and Linux and post processors are available for it including the QtPlasmaC configuration. SheetCam allows you to nest parts over a full sheet of material and allows you to configure toolsets and code snippets to suit your needs. SheetCam post processors are text files written in the Lua programming language and are generally easy to modify to suit your exact requirements. For further information, consult the [SheetCam web site](#) and their support forum.

Another popular post-processor is included with the popular Fusion360 package but the included post-processors will need some customisation.

LinuxCNC is a CNC application and discussions of CAM techniques other than this introductory discussion are out of scope of LinuxCNC.

Chapter 5

User Interfaces

5.1 AXIS GUI

5.1.1 Introduction

AXIS is a graphical front-end for LinuxCNC which features a live preview and backplot. It is written in Python and uses Tk and OpenGL to display its user interface.

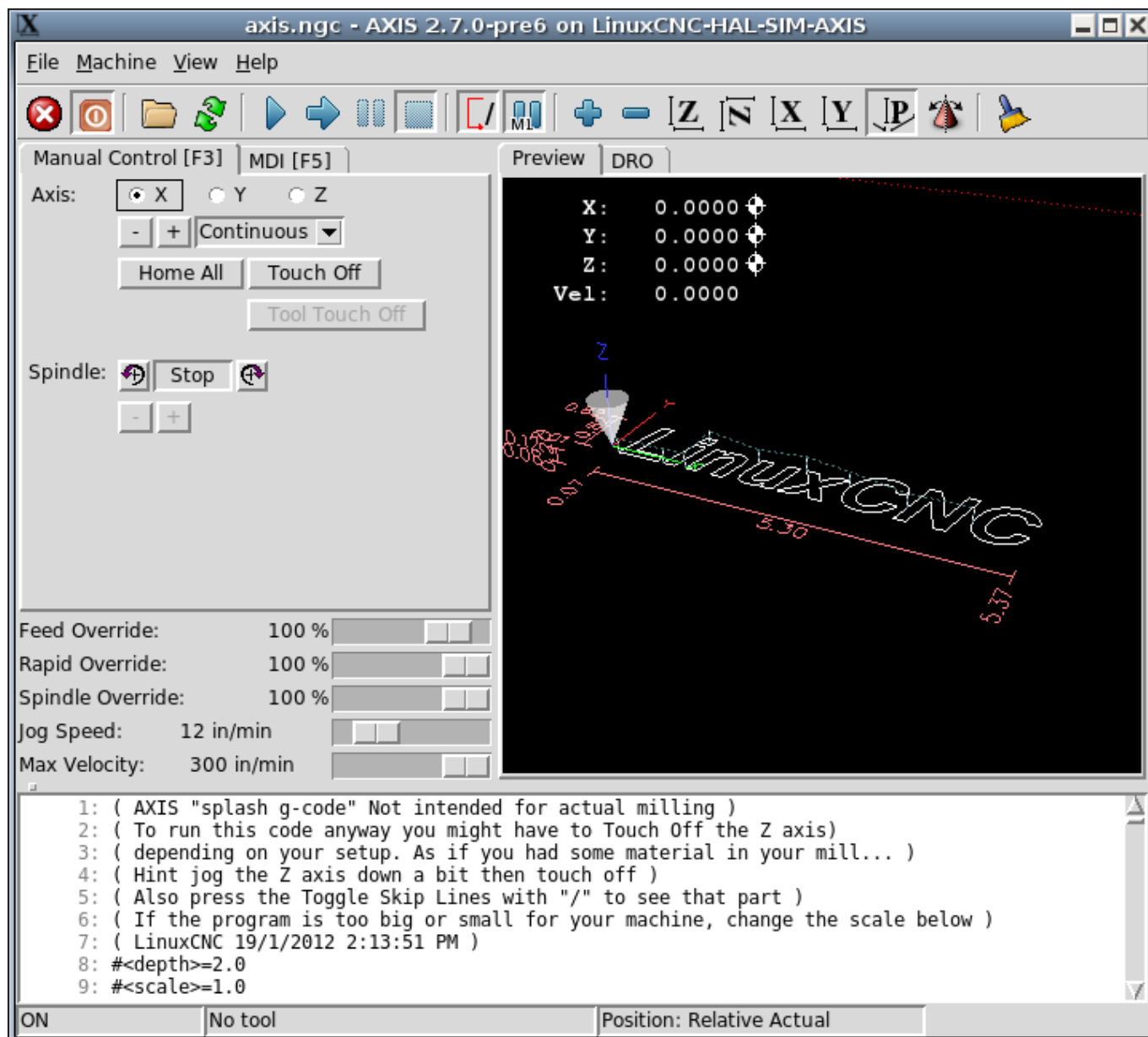


Figure 5.1: The AXIS Window

5.1.2 Getting Started

If your configuration is not currently set up to use AXIS, you can change it by editing the .ini file (INI file). In the section [DISPLAY] change the [DISPLAY] line to read `DISPLAY = axis`.

The sample configuration `sim/axis.ini` is already configured to use AXIS as its front-end.

When AXIS starts, a window like the one in the figure Figure 5.1 above opens.

5.1.2.1 INI settings

For more information on INI file settings that can change how AXIS works see the [Display Section](#) of the INI Configuration Chapter.

- *CYCLE_TIME* - Adjust the response rate of the GUI in milliseconds. Typical 100, useable range 50 - 200
(will accept time in seconds (.05 -.2) for legacy reasons - milliseconds preferred to match other screens).

```
[DISPLAY]
CYCLE_TIME = 100
```

- *PREVIEW_TIMEOUT* - Set timeout in seconds for loading G-code preview. If parsing the G-code lasts longer than this, a notice is shown and only the initial part of the program is drawn in the graphical display. Specifying 0 or leaving the setting out results in no timeout.

```
[DISPLAY]
PREVIEW_TIMEOUT = 5
```

5.1.2.2 A Typical Session

1. Launch LinuxCNC and select a configuration file.
2. Release the E-STOP button (F1) and turn the Machine Power on (F2).
3. Home all axes.
4. Load the G-code file.
5. Use the preview plot to verify that the program is correct.
6. Load the material.
7. Set the proper offset for each axis by jogging and using the Touch Off button as needed.
8. Run the program.
9. To machine the same file again, return to step 6. To machine a different file, return to step 4.
10. When the job is complete, exit AXIS.

Note

To run the same program again depends on your setup and requirements. You might need to load more material and set offsets or move over and set an offset then run the program again. If your material is fixtured then you might need to only run the program again. See the [Machine Menu](#) for more information on the run command.

5.1.3 AXIS Window

The AXIS window contains the following elements:

- A display area that shows one of the following:
 - A preview of the loaded file (in this case, *axis.ngc*), as well as the current location of the CNC machine's *controlled point*. Later, this area will display the path the CNC machine has moved through, called the *backplot*.
 - A large readout showing the current position and all offsets.
-

- A menu bar and toolbar that allow you to perform various actions
- *Manual Control Tab* - which allows you to make the machine move, turn the spindle on or off, and turn the coolant on or off if included in the INI file.
- *MDI Tab* - where G-code programs can be entered manually, one line at a time. This also shows the *Active G-codes* which shows which modal G-codes are in effect.
- *Feed Override* - which allows you to scale the speed of programmed motions. The default maximum is 120% and can be set to a different value in the INI file. See the [Display Section](#) of the INI file for more information.
- *Spindle Override* - which allows you to scale the spindle speed up or down.
- *Jog Speed* - which allows you to set the jog speed within the limits set in the INI file. See the [Display Section](#) of the INI file for more information.
- *Max Velocity* - which allows you to restrict the maximum velocity of all programmed motions (except spindle synchronized motion).
- A text display area that shows the loaded G-code.
- A status bar which shows the state of the machine. In this screen shot, the machine is turned on, does not have a tool inserted, and the displayed position is *Relative* (showing all offsets), and *Actual* (showing feedback position).

5.1.3.1 Menu Items

Some menu items might be grayed out depending on how you have your INI file configured. For more information on configuration see the [INI Chapter](#).

- *Open...* - Opens a standard dialog box to open a G-code file to load in AXIS. If you have configured LinuxCNC to use a filter program you can also open it up. See the [FILTER Section](#) of the INI configuration for more information.
 - *Recent Files* - Displays a list of recently opened files.
 - *Edit...* - Open the current G-code file for editing if you have an editor configured in your INI file. See the [DISPLAY Section](#) for more information on specifying an editor to use.
 - *Reload* - Reload the current G-code file. If you edited it you must reload it for the changes to take affect. If you stop a file and want to start from the beginning then reload the file. The toolbar reload is the same as the menu.
 - *Save G-code as...* - Save the current file with a new name.
 - *Properties* - The sum of the rapid and feed moves. Does not factor in acceleration, blending or path mode so time reported will never be less than the actual run time.
 - *Edit tool table...* - Same as Edit if you have defined an editor you can open the tool table and edit it.
 - *Reload tool table* - After editing the tool table you must reload it.
 - *Ladder editor* - If you have loaded ClassicLadder you can edit it from here. See the [ClassicLadder chapter](#) for more information.
 - *Quit* - Terminates the current LinuxCNC session.
 - *Toggle Emergency Stop F1* - Change the state of the Emergency Stop.
 - *Toggle Machine Power F2* - Change the state of the Machine Power if the Emergency Stop is not on.
-

- *Run Program* - Run the currently loaded program from the beginning.
- *Run From Selected Line* - Select the line you want to start from first. Use with caution as this will move the tool to the expected position before the line first then it will execute the rest of the code.

**Warning**

Do not use *Run From Selected Line* if your G-code program contains subroutines.

- *Step* - Single step through a program.
 - *Pause* - Pause a program.
 - *Resume* - Resume running from a pause.
 - *Stop* - Stop a running program. When run is selected after a stop the program will start from the beginning.
 - *Stop at M1* - If an M1 is reached, and this is checked, program execution will stop on the M1 line. Press Resume to continue.
 - *Skip lines with "/"* - If a line begins with / and this is checked, the line will be skipped.
 - *Clear MDI history* - Clears the MDI history window.
 - *Copy from MDI history* - Copies the MDI history to the clipboard
 - *Paste to MDI history* - Paste from the clipboard to the MDI history window
 - *Calibration* - Starts the Calibration assistant (emccalib.tcl). Calibration reads the HAL file and for every *setp* that uses a variable from the INI file that is in an [AXIS_L],[JOINT_N],[SPINDLE_S], or [TUNE] section it creates an entry that can be edited and tested.
 - *Show HAL Configuration* - Opens the HAL Configuration window where you can monitor HAL Components, Pins, Parameters, Signals, Functions, and Threads.
 - *HAL Meter* - Opens a window where you can monitor a single HAL Pin, Signal, or Parameter.
 - *HAL Scope* - Opens a virtual oscilloscope that allows plotting HAL values vs. time.
 - *Show LinuxCNC Status* - Opens a window showing LinuxCNC's status.
 - *Set Debug Level* - Opens a window where debug levels can be viewed and some can be set.
 - *Homing* - Home one or all axes.
 - *Unhoming* - Unhome one or all axes.
 - *Zero Coordinate System* - Set all offsets to zero in the coordinate system chosen.
 - Tool Touch Off
 - *Tool touch off to workpiece* - When performing Touch Off, the value entered is relative to the current workpiece (G5x) coordinate system, as modified by the axis offset (G92). When the Touch Off is complete, the Relative coordinate for the chosen axis will become the value entered. See [G10 L10](#) in the G-code chapter.
 - *Tool touch off to fixture* - When performing Touch Off, the value entered is relative to the ninth (G59.3) coordinate system, with the axis offset (G92) ignored. This is useful when there is a tool touch-off fixture at a fixed location on the machine, with the ninth (G59.3) coordinate system set such that the tip of a zero-length tool is at the fixture's origin when the Relative coordinates are 0. See [G10 L11](#) in the G-code chapter.
-

- *Top View* - The Top View (or Z view) displays the G-code looking along the Z axis from positive to negative. This view is best for looking at X & Y.
- *Rotated Top View* - The Rotated Top View (or rotated Z view) also displays the G-code looking along the Z axis from positive to negative. But sometimes it's convenient to display the X & Y axes rotated 90 degrees to fit the display better. This view is also best for looking at X & Y.
- *Side View* - The Side View (or X view) displays the G-code looking along the X axis from positive to negative. This view is best for looking at Y & Z.
- *Front View* - The Front View (or Y view) displays the G-code looking along the Y axis from negative to positive. This view is best for looking at X & Z.
- *Perspective View* - The Perspective View (or P view) displays the G-code looking at the part from an adjustable point of view, defaulting to X+, Y-, Z+. The position is adjustable using the mouse and the drag/rotate selector. This view is a compromise view, and while it does do a good job of trying to show three (to nine!) axes on a two-dimensional display, there will often be some feature that is hard to see, requiring a change in viewpoint. This view is best when you would like to see all three (to nine) axes at once.

Point of View

The AXIS display pick menu *View* refers to *Top*, *Front*, and *Side* views. These terms are correct if the CNC machine has its Z axis vertical, with positive Z up. This is true for vertical mills, which is probably the most popular application, and also true for almost all EDM machines, and even vertical turret lathes, where the part is turning below the tool.

The terms *Top*, *Front*, and *Side* might be confusing in other CNC machines, such as a standard lathe, where the Z axis is horizontal, or a horizontal mill, again where the Z axis is horizontal, or even an inverted vertical turret lathe, where the part is turning above the tool, and the Z axis positive direction is down!








Just remember that positive Z axis is (almost) always away from the part. So be familiar with your machine's design and interpret the display as needed.













- *Display Inches* - Set the AXIS display scaling for inches.
- *Display MM* - Set the AXIS display scaling for millimeters.
- *Show Program* - The preview display of the loaded G-code program can be entirely disabled if desired.
- *Show Program Rapids* - The preview display of the loaded G-code program will always show the feed rate moves (G1,G2,G3) in white. But the display of rapid moves (G0) in cyan can be disabled if desired.
- *Alpha-blend Program* - This option makes the preview of complex programs easier to see, but may cause the preview to display more slowly.
- *Show Live Plot* - The highlighting of the feedrate paths (G1,G2,G3) as the tool moves can be disabled if desired.
- *Show Tool* - The display of the tool cone/cylinder can be disabled if desired.
- *Show Extents* - The display of the extents (maximum travel in each axis direction) of the loaded G-code program can be disabled if desired.
- *Show Offsets* - The selected fixture offset (G54-G59.3) origin location can be shown as a set of three orthogonal lines, one each of red, blue, and green. This offset origin (or fixture zero) display can be disabled if desired.

- *Show Machine Limits* - The machine's maximum travel limits for each axis, as set in the INI file, are shown as a rectangular box drawn in red dashed lines. This is useful when loading a new G-code program, or when checking for how much fixture offset would be needed to bring the G-code program within the travel limits of your machine. It can be shut off if not needed.
- *Show Velocity* - A display of velocity is sometimes useful to see how close your machine is running to its design velocities. It can be disabled if desired.
- *Show Distance to Go* - Distance to go is a very handy item to know when running an unknown G-code program for the first time. In combination with the rapid override and feedrate override controls, unwanted tool and machine damage can be avoided. Once the G-code program has been debugged and is running smoothly, the Distance to Go display can be disabled if desired.
- *Coordinates in large font...* - The coordinates of the axes and the speed in advance, will display in large font in the toolpath view.
- *Clear Live Plot* - As the tool travels in the AXIS display, the G-code path is highlighted. To repeat the program, or to better see an area of interest, the previously highlighted paths can be cleared.
- *Show Commanded Position* - This is the position that LinuxCNC will try to go to. Once motion has stopped, this is the position LinuxCNC will try to hold.
- *Show Actual Position* - Actual Position is the measured position as read back from the system's encoders or simulated by step generators. This may differ slightly from the Commanded Position for many reasons including PID tuning, physical constraints, or position quantization.
- *Show Machine Position* - This is the position in unoffset coordinates, as established by Homing.
- *Show Relative Position* - This is the Machine Position modified by G5x, G92, and G43 offsets.
- *About AXIS* - We all know what this is.
- *Quick Reference* - Shows the keyboard shortcut keys.


5.1.3.2 Toolbar buttons


From left to right in the AXIS display, the toolbar buttons (keyboard shortcuts shown [in brackets]) are:

-  Toggle Emergency Stop [F1] (also called E-Stop)
-  Toggle Machine Power [F2]
-  Open G Code file [O]
-  Reload current file [Ctrl-R]
-  Begin executing the current file [R]
-  Execute next line [T]
-  Pause Execution [P] Resume Execution [S]


-  Stop Program Execution [ESC]
-  Toggle Skip lines with "/" [Alt-M-/]
-  Toggle Optional Pause [Alt-M-1]
-  Zoom In
-  Zoom Out
-  Top view
-  Rotated Top view
-  Side view
-  Front view
-  Perspective view
-  Toggle between Drag and Rotate Mode [D]
-  Clear live backplot [Ctrl-K]

5.1.3.3 Graphical Display Area

Coordinate Display In the upper-left corner of the program display is the coordinate position display for each axis. To the right of the number an origin symbol  is shown if the axis has been homed.

A limit symbol  is shown on the right side of the coordinate position number if the axis is on one of its limit switches.

To properly interpret the coordinate position numbers, refer to the *Position:* indicator in the status bar. If the position is *Machine Actual*, then the displayed number is in the machine coordinate system. If it is *Relative Actual*, then the displayed number is in the offset coordinate system. When the coordinates

displayed are relative and an offset has been set, the display will include a cyan [machine origin](#)  marker.

If the position is *Commanded*, then the exact coordinate given in a G-code command is displayed. If it is *Actual*, then it is the position the machine has actually moved to. These values can be different from commanded position due to following error, dead band, encoder resolution, or step size. For instance, if you command a movement to X 0.0033 on your mill, but one step of your stepper motor or one encoder count is 0.00125, then the *Commanded* position might be 0.0033, but the *Actual* position will be 0.0025 (2 steps) or 0.00375 (3 steps).

Preview Plot When a file is loaded, a preview of it is shown in the display area. Fast moves (such as those produced by the G0 command) are shown as cyan lines. Moves at a feed rate (such as those

produced by the *G1* command) are shown as solid white lines. Dwells (such as those produced by the *G4* command) are shown as small pink X marks.

G0 (Rapid) moves prior to a feed move will not show on the preview plot. Rapid moves after a T<n> (Tool Change) will not show on the preview until after the first feed move. To turn either of these features off program a G1 without any moves prior to the G0 moves.

Program Extents The *extents* of the program in each axis are shown. At the ends, the least and greatest coordinate values are indicated. In the middle, the difference between the coordinates is shown.

When some coordinates exceed the *soft limits* in the INI file, the relevant dimension is shown in a different color and enclosed by a box. In figure below the maximum soft limit is exceeded on the X axis as indicated by the box surrounding the coordinate value. The minimum X travel of the program is -1.95, the maximum X travel is 1.88, and the program requires 3.83 inches of X travel. To move the program so it's within the machine's travel in this case, jog to the left and Touch Off X again.



Figure 5.2: Soft Limits

Tool Cone When no tool is loaded, the location of the tip of the tool is indicated by the *tool cone*. The *tool cone* does not provide guidance on the form, length, or radius of the tool.

When a tool is loaded (for instance, with the MDI command *T1 M6*), the cone changes to a cylinder which shows the diameter of the tool given in the tool table file.

Backplot When the machine moves, it leaves a trail called the backplot. The color of the line indicates the type of motion: Yellow for jogs, faint green for rapid movements, red for straight moves at a feed rate, and magenta for circular moves at a feed rate.

Grid AXIS can optionally display a grid when in orthogonal views. Enable or disable the grid using the *Grid* menu under *View*. When enabled, the grid is shown in the top and rotated top views; when coordinate system is not rotated, the grid is shown in the front and side views as well. The presets in the *Grid* menu are controlled by the INI file item [DISPLAY]GRIDS. If unspecified, the default is 10mm 20mm 50mm 100mm 1in 2in 5in 10in.

Specifying a very small grid may decrease performance.

Interacting By left-clicking on a portion of the preview plot, the line will be highlighted in both the graphical and text displays. By left-clicking on an empty area, the highlighting will be removed.

By dragging with the left mouse button pressed, the preview plot will be shifted (panned).

By dragging with shift and the left mouse button pressed, or by dragging with the mouse wheel pressed, the preview plot will be rotated. When a line is highlighted, the center of rotation is the center of the line. Otherwise, the center of rotation is the center of the entire program.

By rotating the mouse wheel, or by dragging with the right mouse button pressed, or by dragging with control and the left mouse button pressed, the preview plot will be zoomed in or out.

By clicking one of the *Preset View* icons, or by pressing *V*, several preset views may be selected.

5.1.3.4 Text Display Area

By left-clicking a line of the program, the line will be highlighted in both the graphical and text displays.

When the program is running, the line currently being executed is highlighted in red. If no line has been selected by the user, the text display will automatically scroll to show the current line.

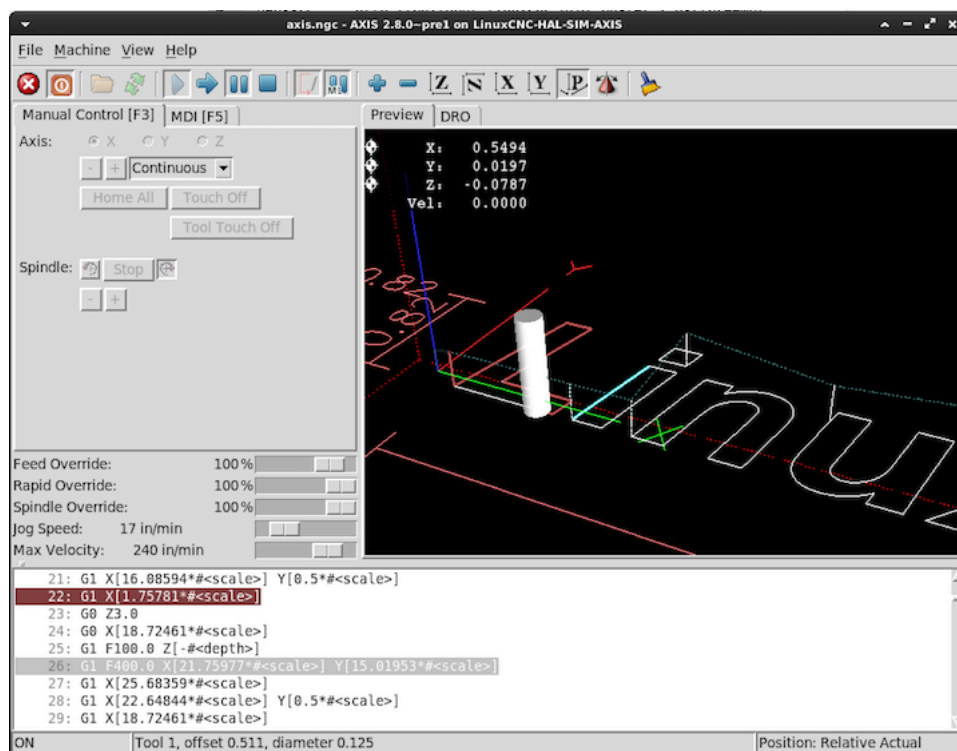


Figure 5.3: Current and Selected Lines

5.1.3.5 Manual Control

While the machine is turned on but not running a program, the items in the *Manual Control* tab can be used to move the machine or control its spindle and coolant.

When the machine is not turned on, or when a program is running, the manual controls are unavailable.

Many of the items described below are not useful on all machines. When AXIS detects that a particular pin is not connected in HAL, the corresponding item in the Manual Control tab is removed. For instance, if the HAL pin *spindle.0.brake* is not connected, then the *Brake* button will not appear on the screen. If the environment variable *AXIS_NO_AUTOCONFIGURE* is set, this behavior is disabled and all the items will appear.

The Axis group AXIS allows you to manually move the machine. This action is known as *jogging*. First, select the axis to be moved by clicking it. Then, click and hold the + or - button depending on the desired direction of motion. The first four axes can also be moved by the arrow keys (X and Y), PAGE UP and PAGE DOWN keys (Z), and the [and] keys (A).

If *Continuous* is selected, the motion will continue as long as the button or key is pressed. If another value is selected, the machine will move exactly the displayed distance each time the button is clicked or the key is pressed. By default, the available values are 0.1000, 0.0100, 0.0010, 0.0001.

See the [DISPLAY Section](#) for more information on setting the increments.

Homing (Identity Kinematics) The INI file setting [KINS]JOINTS defines the total number of joints for the system. A joint may be configured with a home switch or for *immediate* homing. Joints may specify a home sequence that organizes the order for homing groups of joints.

If **all** joints are configured for homing and have valid home sequences, the homing button will show *Home All*. Pressing the *Home All* button (or the Ctrl-HOME key) will initiate homing for all joints using their defined home sequences. Pressing the HOME key will home the joint corresponding to the currently selected axis even if no homing sequence is defined.

If not all axes have valid home sequences, the homing button will show *Home Axis* and will home the joint for the currently selected axis only. Each axis must be selected and homed separately.

The dropdown menu Machine/Homing provides an alternate method to home axes. The dropdown menu Machine/Unhoming provides means to unhom home axes.

If your machine does not have home switches defined in the configuration, the *Home* button will set the current position of the selected axis as the absolute position 0 for that axis and will set the *is-homed* bit for that axis.

See the [Homing Configuration Chapter](#) for more information.

Homing (Non-Identity Kinematics) Operation is similar to that for Identity Kinematics but, prior to homing, the selection radio buttons select joints by number. The homing button will show *Home All* if all joints are configured for homing and have valid home sequences. Otherwise, the homing button will show *Home Joint*.

See the [Homing Configuration Chapter](#) for more information.

Touch Off

By pressing *Touch Off* or the END key, the *G5x offset* for the current axis is changed so that the current axis value will be the specified value. Expressions may be entered using the rules for rs274ngc programs, except that variables may not be referred to. The resulting value is shown as a number.

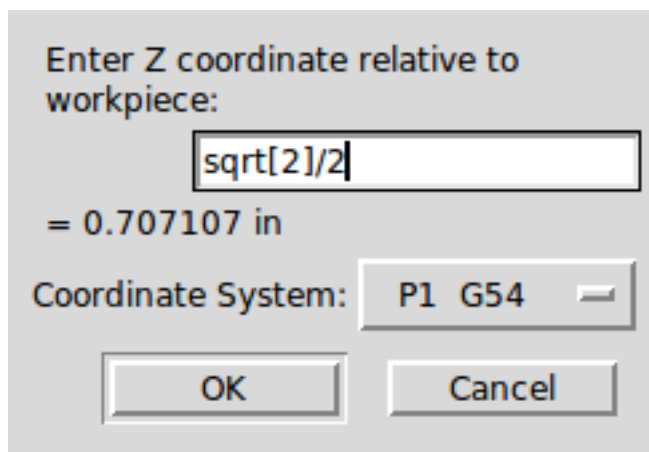


Figure 5.4: Touch Off Window

See also the Machine menu options: *Touch part* and *Touch part holder*.

Tool Touch Off By pressing the *Tool Touch Off* button the tool length and offsets of the currently loaded tool will be changed so that the current tool tip position matches the entered coordinate.

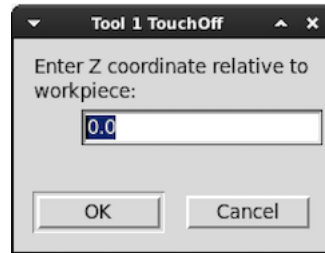


Figure 5.5: Tool Touch Off Window

See also the *Tool touch off to workpiece* and *Tool touch off to fixture* options in the Machine menu.

Override Limits By pressing Override Limits, the machine will temporarily be allowed to jog off of a physical limit switch. This check box is only available when a limit switch is tripped. The override is reset after one jog. If the axis is configured with separate positive and negative limit switches, LinuxCNC will allow the jog only in the correct direction. *Override Limits will not allow a jog past a soft limit. The only way to disable a soft limit on an axis is to Unhome it.*

The Spindle group The buttons on the first row select the direction for the spindle to rotate: Counterclockwise, Stopped, Clockwise. Counterclockwise will only show up if the pin *spindle.0.reverse* is in the HAL file (it can be *net trick-axis spindle.0.reverse*). The buttons on the next row increase or decrease the rotation speed. The checkbox on the third row allows the spindle brake to be engaged or released. Depending on your machine configuration, not all the items in this group may appear. Pressing the spindle start button sets the S speed to 1.

The Coolant group The two buttons allow the *Mist* and *Flood* coolants to be turned on and off. Depending on your machine configuration, not all the items in this group may appear.

5.1.3.6 MDI

MDI allows G-code commands to be entered manually. When the machine is not turned on, or when a program is running, the MDI controls are unavailable.

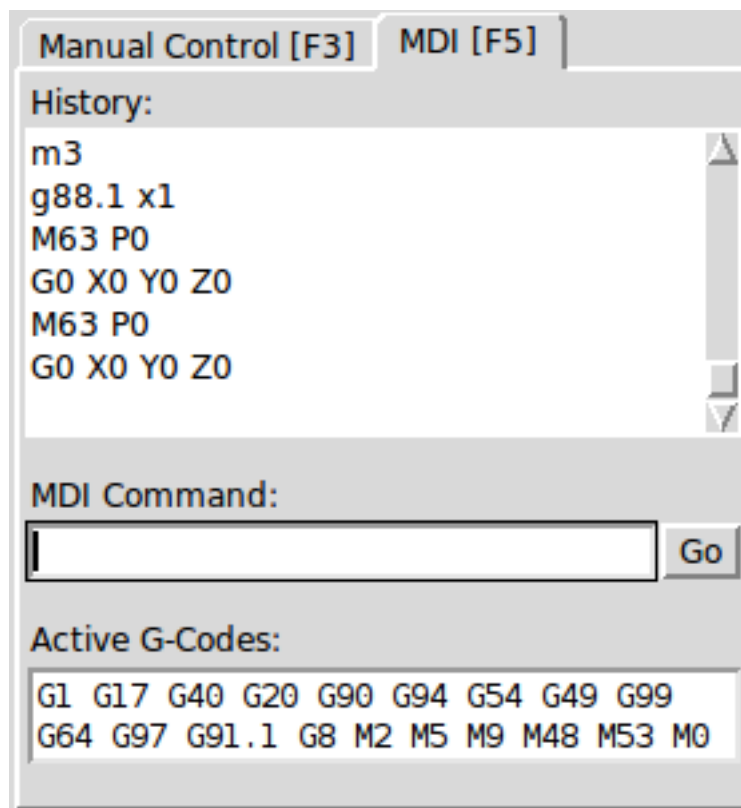


Figure 5.6: The MDI tab

- *History* - This shows MDI commands that have been typed earlier in this session.
- *MDI Command* - This allows you to enter a G-code command to be executed. Execute the command by pressing Enter or by clicking Go.
- *Active G-codes* - This shows the *modal codes* that are active in the interpreter. For instance, *G54* indicates that the *G54 offset* is applied to all coordinates that are entered. When in Auto the Active G-codes represent the codes after any read ahead by the interpreter.

5.1.3.7 Feed Override

By moving this slider, the programmed feed rate can be modified. For instance, if a program requests *F60* and the slider is set to 120%, then the resulting feed rate will be 72.

5.1.3.8 Spindle Speed Override

By moving this slider, the programmed spindle speed can be modified. For instance, if a program requests *S8000* and the slider is set to 80%, then the resulting spindle speed will be 6400. This item only appears when the HAL pin *spindle.0.speed-out* is connected.

5.1.3.9 Jog Speed

By moving this slider, the speed of jogs can be modified. For instance, if the slider is set to 1 in/min, then a .01 inch jog will complete in about .6 seconds, or 1/100 of a minute. Near the left side (slow

jogs) the values are spaced closely together, while near the right side (fast jogs) they are spaced much further apart, allowing a wide range of jog speeds with fine control when it is most important.

On machines with a rotary axis, a second jog speed slider is shown. This slider sets the jog rate for the rotary axes (A, B and C).

5.1.3.10 Max Velocity

By moving this slider, the maximum velocity can be set. This caps the maximum velocity for all programmed moves except spindle-synchronized moves.

5.1.4 Keyboard Controls

Almost all actions in AXIS can be accomplished with the keyboard. A full list of keyboard shortcuts can be found in the AXIS Quick Reference, which can be displayed by choosing Help > Quick Reference. Many of the shortcuts are unavailable when in MDI mode.

5.1.4.1 Feed Override Keys

Note

For details on the Spanish keyboard layout please inspect the translated documentation.

The Feed Override keys behave differently when in Manual Mode. The keys '12345678 will select an axis if it is programmed. If you have 3 axis then ' will select axis 0, 1 will select axis 1, and 2 will select axis 2. The remainder of the number keys will still set the Feed Override. When running a program '1234567890 will set the Feed Override to 0% - 100%.

The most frequently used keyboard shortcuts are shown in the following table:

Table 5.1: Most Common Keyboard Shortcuts

Keystroke	Action Taken	Mode
F1	Toggle Emergency Stop	Any
F2	Turn machine on/off	Any
`, 1 .. 9, 0	Set feed override from 0% to 100%	Varies
X, `	Activate first axis	Manual
Y, 1	Activate second axis	Manual
Z, 2	Activate third axis	Manual
A, 3	Activate fourth axis	Manual
I	Select jog increment	Manual
C	Continuous jog	Manual
Control-Home	Perform homing sequence	Manual
End	Touch off: Set G5x offset for active axis	Manual
Left, Right	Jog first axis	Manual
Up, Down	Jog second axis	Manual
Pg Up, Pg Dn	Jog third axis	Manual
[,]	Jog fourth axis	Manual
O	Open File	Manual
Control-R	Reload File	Manual

Table 5.1: (continued)

Keystroke	Action Taken	Mode
R	Run file	Manual
P	Pause execution	Auto
S	Resume Execution	Auto
ESC	Stop execution	Auto
Control-K	Clear backplot	Auto/Manual
V	Cycle among preset views	Auto/Manual
Shift-Left,Right	Rapid X Axis	Manual
Shift-Up,Down	Rapid Y Axis	Manual
Shift-PgUp, PgDn	Rapid Z Axis	Manual
@	toggle Actual/Commanded	Any
#	toggle Relative/Machine	Any

5.1.5 Show LinuxCNC Status (linuxcnc top)

AXIS includes a program called *linuxcnc top* which shows some of the details of LinuxCNC's state. You can run this program by invoking Machine > Show LinuxCNC Status

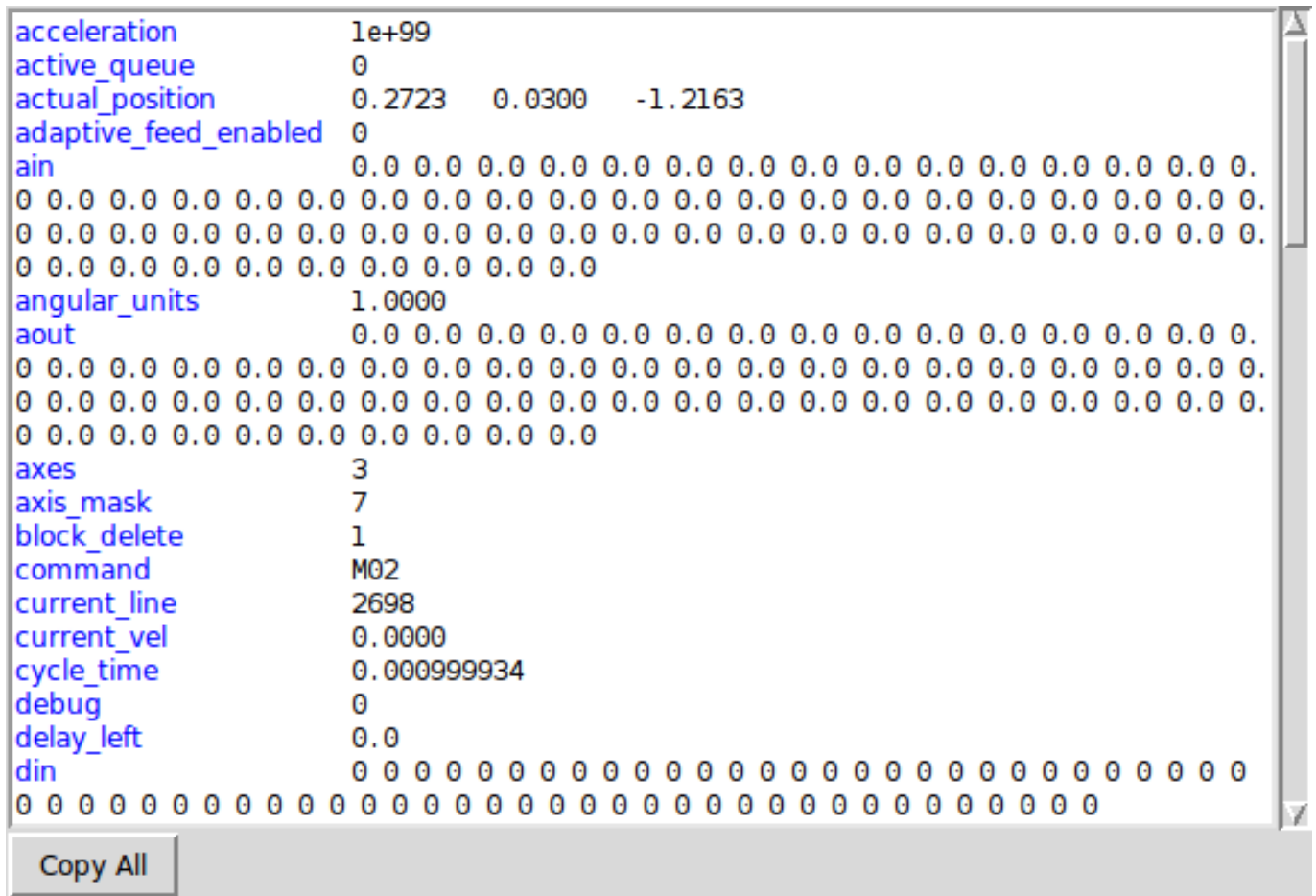


Figure 5.7: LinuxCNC Status Window

The name of each item is shown in the left column. The current value is shown in the right column. If the value has recently changed, it is shown on a red background.

5.1.6 MDI interface

AXIS includes a program called `mdi` which allows text-mode entry of MDI commands to a running LinuxCNC session. You can run this program by opening a terminal and typing:

mdi

Once it is running, it displays the prompt *MDI>*. When a blank line is entered, the machine's current position is shown. When a command is entered, it is sent to LinuxCNC to be executed.

This is a sample session of mdi:

```
$ mdi
MDI>
(0.0, 0.0, 0.0, 0.0, 0.0, 0.0)
MDI> G1 F5 X1
MDI>
(0.59285000000000374, 0.0, 0.0, 0.0, 0.0, 0.0)
MDI>
(1.00000000000000639, 0.0, 0.0, 0.0, 0.0, 0.0)
```

5.1.7 axis-remote

AXIS includes a program called *axis-remote* which can send certain commands to a running AXIS. The available commands are shown by running *axis-remote --help* and include checking whether AXIS is running (*--ping*), loading a file by name, reloading the currently loaded file (*--reload*), and making AXIS exit (*--quit*).

5.1.8 Manual Tool Change

LinuxCNC includes a userspace HAL component called *hal_manualtoolchange*, which shows a window prompt telling you what tool is expected when a *M6* command is issued. After the OK button is pressed, execution of the program will continue.

The *hal_manualtoolchange* component includes a HAL pin for a button that can be connected to a physical button to complete the tool change and remove the window prompt (*hal_manualtoolchange.change_button*).

The HAL configuration file *lib/hallib/axis_manualtoolchange.hal* shows the HAL commands necessary to use this component.

hal_manualtoolchange can be used even when AXIS is not used as the GUI. This component is most useful if you have presettable tools and you use the tool table.

Note

Important Note: Rapids will not show on the preview after a *T<n>* is issued until the next feed move after the *M6*. This can be very confusing to most users. To turn this feature off for the current tool change program a *G1* with no move after the *T<n>*.

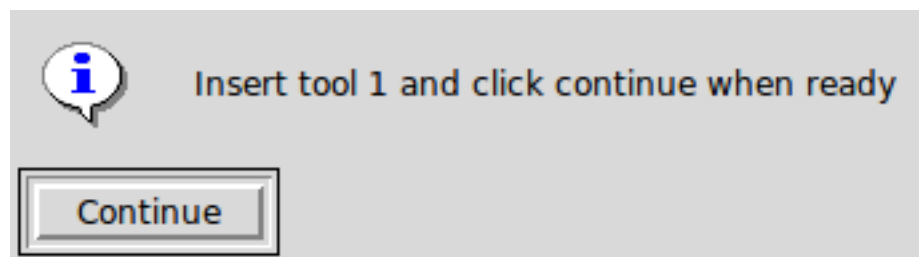


Figure 5.8: Manual Toolchange Window

5.1.9 Python modules

AXIS includes several Python modules which may be useful to others. For more information on one of these modules, use *pydoc <module name>* or read the source code. These modules include:

- *emc* provides access to the LinuxCNC command, status, and error channels
- *gcode* provides access to the rs274ngc interpreter
- *rs274* provides additional tools for working with rs274ngc files
- *hal* allows the creation of userspace HAL components written in Python
- *_togl* provides an OpenGL widget that can be used in Tkinter applications
- *minigl* provides access to the subset of OpenGL used by AXIS

To use these modules in your own scripts, you must ensure that the directory where they reside is on Python's module path. When running an installed version of LinuxCNC, this should happen automatically. When running *in-place*, this can be done by using *scripts/rip-environment*.

5.1.10 Using AXIS in Lathe Mode

By including the line `LATHE = 1` in the [DISPLAY] section of the INI file, AXIS selects lathe mode. The Y axis is not shown in coordinate readouts, the view is changed to show the Z axis extending to the right and the X axis extending towards the bottom of the screen, and several controls (such as those for preset views) are removed. The coordinate readouts for X are replaced with diameter and radius.

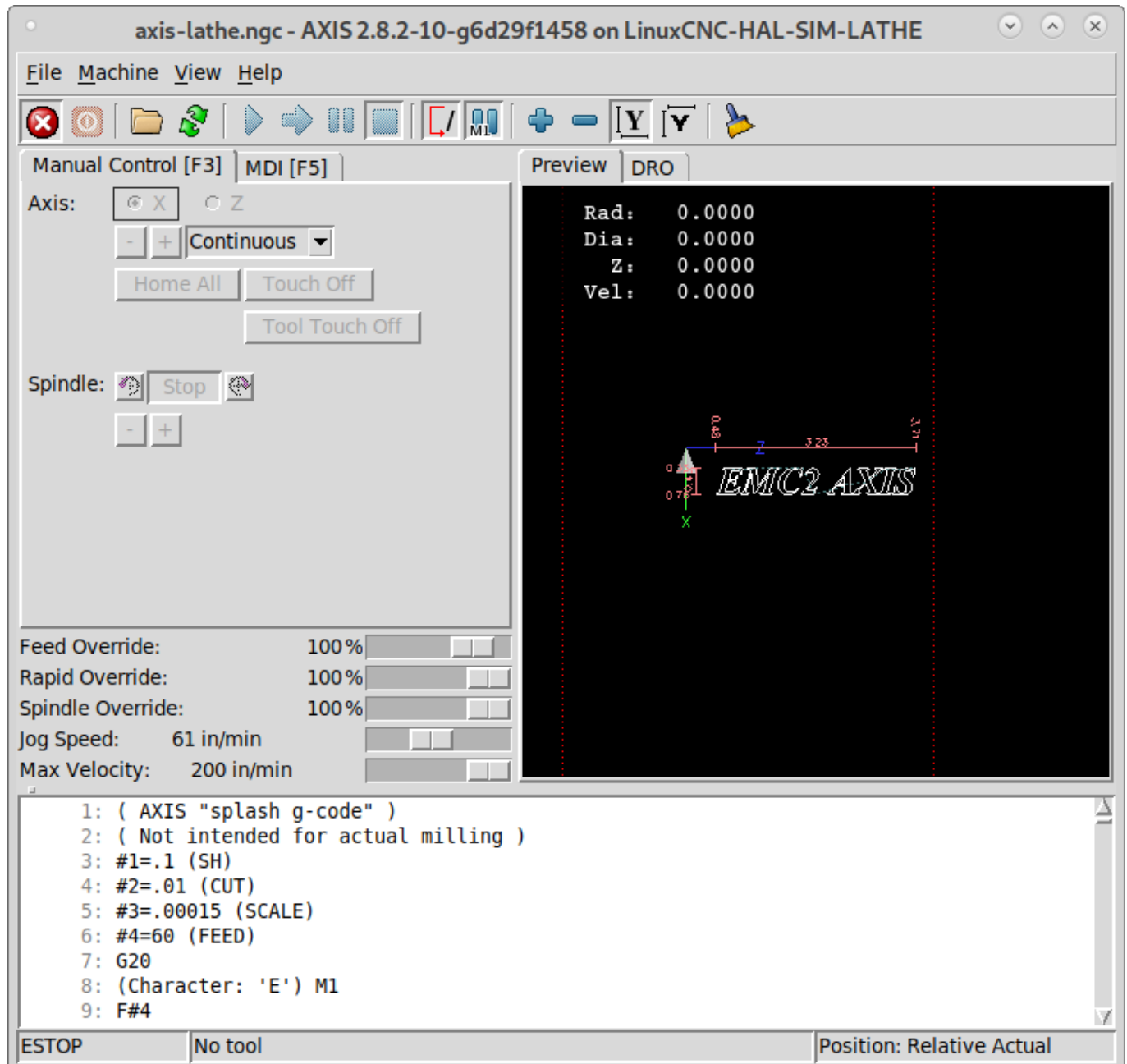


Figure 5.9: AXIS Lathe Mode

Pressing `V` zooms out to show the entire file, if one is loaded.

When in lathe mode, the shape of the loaded tool (if any) is shown.

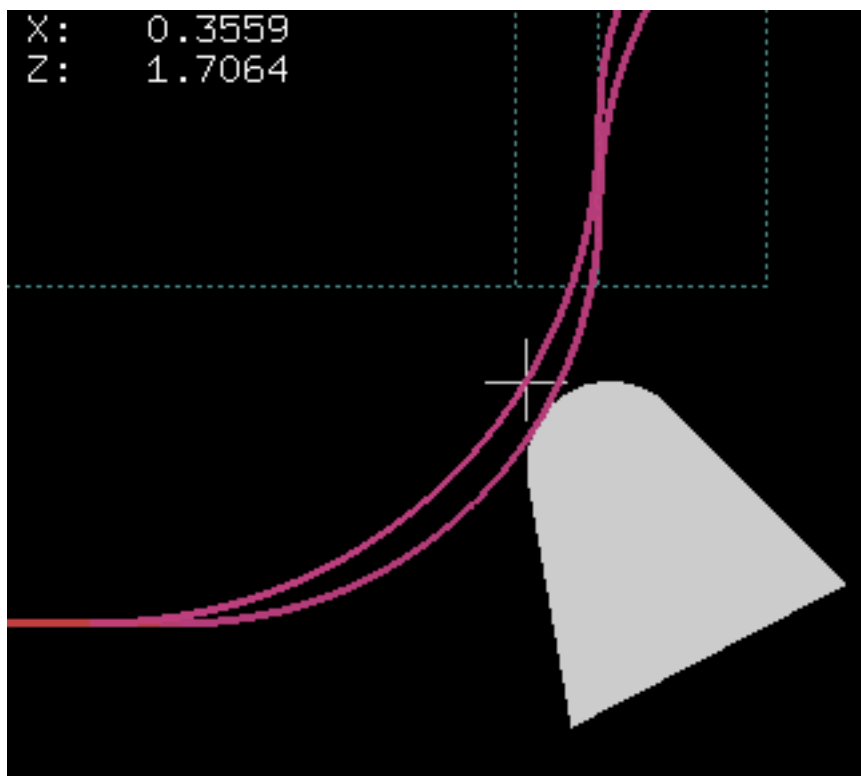


Figure 5.10: Lathe Tool Shape

To change the display to a back tool lathe you need to have both *LATHE* = 1 and *BACK_TOOL_LATHE* = 1 in the [DISPLAY] section. This will invert the view and put the tool on the back side of the Z axis.

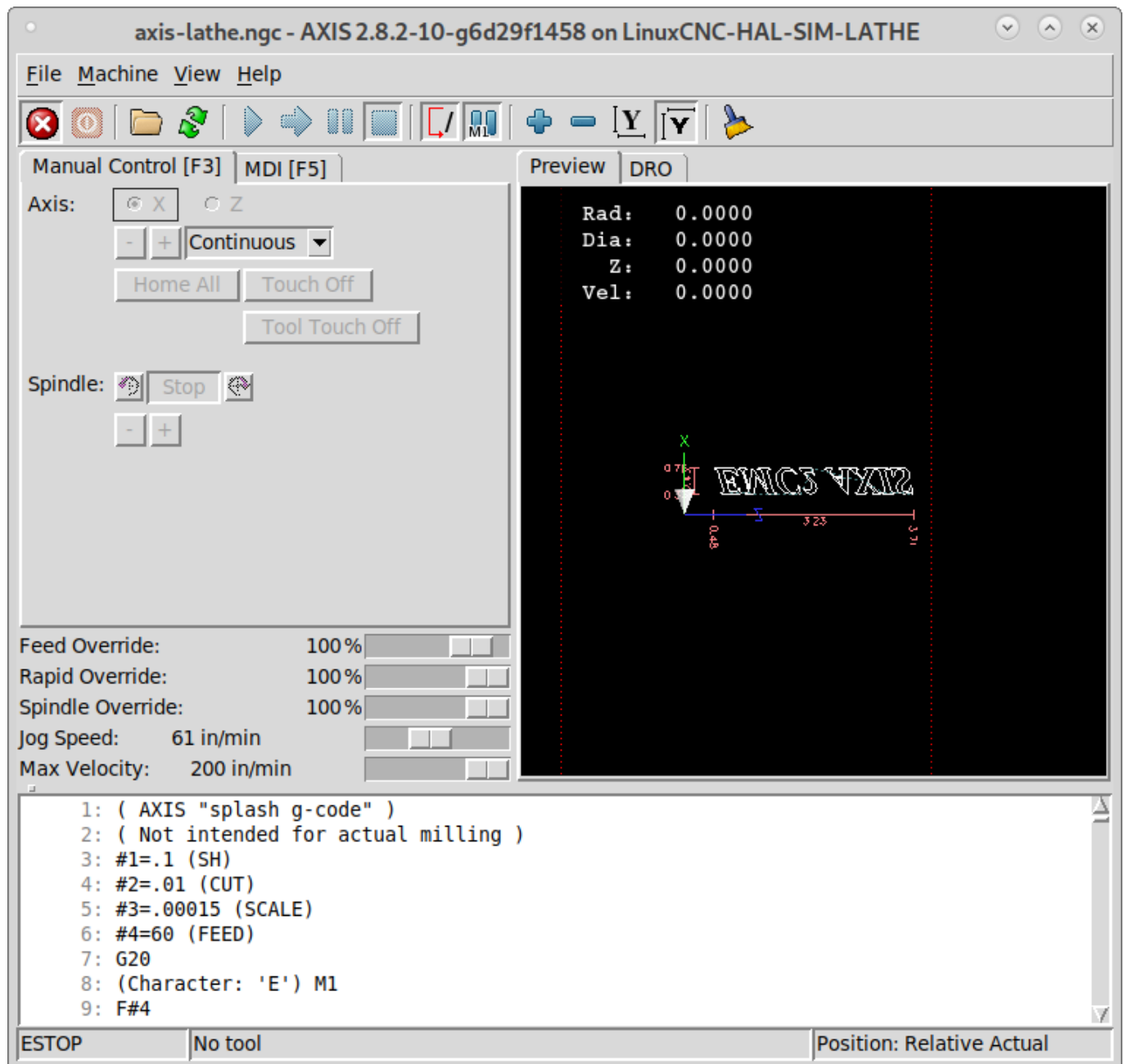


Figure 5.11: Lathe Back Tool Shape

5.1.11 Using AXIS in Foam Cutting mode

By including the line *FOAM = 1* in the [DISPLAY] section of the INI file, AXIS selects foam-cutting mode. In the program preview, XY motions are displayed in one plane, and UV motions in another. In the live plot, lines are drawn between corresponding points on the XY plane and the UV plane. The special comments (XY_Z_POS) and (UV_Z_POS) set the Z coordinates of these planes, which default to 0 and 1.5 machine units.

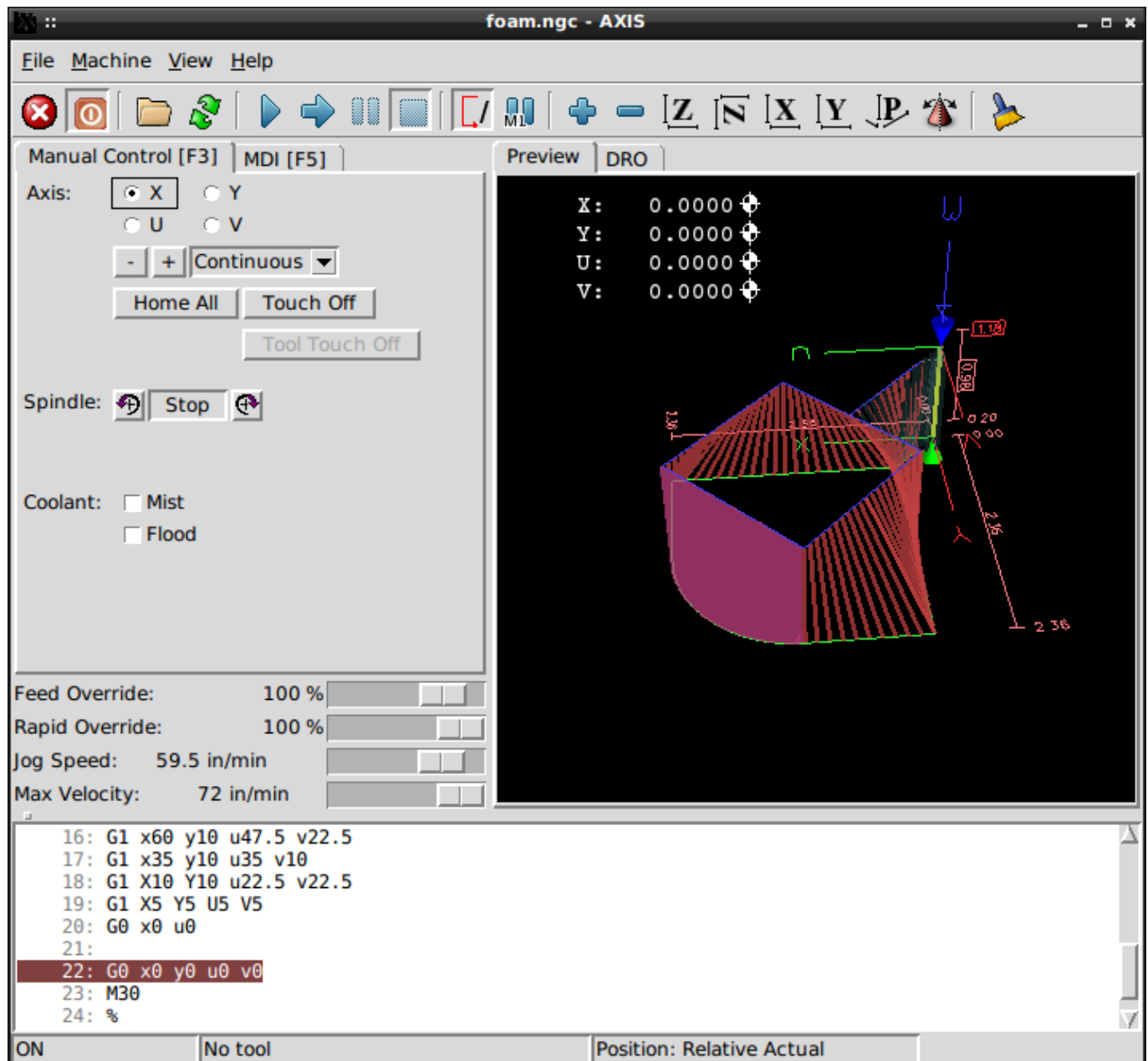


Figure 5.12: Foam Cutting Mode

5.1.12 Advanced Configuration

When AXIS is started it creates the HAL pins for the GUI then it executes the HAL file named in the INI file: `[HAL]POSTGUI_HALFILE=<filename>`. Typically `<filename>` would be the configs base name + `_postgui` + `.hal` eg. `lathe_postgui.hal`, but can be any legal filename. These commands are executed after the screen is built, guaranteeing the widget's HAL pins are available. You can have multiple line of `POSTGUI_HALFILE=<filename>` in the INI. Each will be run one after the other in the order they appear.

For more information on the INI file settings that can change the way AXIS works, see the `<<sub:ini:sec:display,Display Section>>` of the INI configuration chapter.

5.1.12.1 Program Filters

AXIS has the ability to send loaded files through a *filter program*. This filter can do any desired task: Something as simple as making sure the file ends with *M2*, or something as complicated as generating G-code from an image.

The *[FILTER]* section of the INI file controls how filters work. First, for each type of file, write a *PROGRAM_EXTENSION* line. Then, specify the program to execute for each type of file. This program is given the name of the input file as its first argument, and must write rs274ngc code to standard output. This output is what will be displayed in the text area, previewed in the display area, and executed by LinuxCNC when *Run*. The following lines add support for the *image-to-gcode* converter included with LinuxCNC:

```
[FILTER]
PROGRAM_EXTENSION = .png,.gif Greyscale Depth Image
png = image-to-gcode
gif = image-to-gcode
```

It is also possible to specify an interpreter:

```
PROGRAM_EXTENSION = .py Python Script
py = python
```

In this way, any Python script can be opened, and its output is treated as G-code. One such example script is available at *nc_files/holecircle.py*. This script creates G-code for drilling a series of holes along the circumference of a circle.

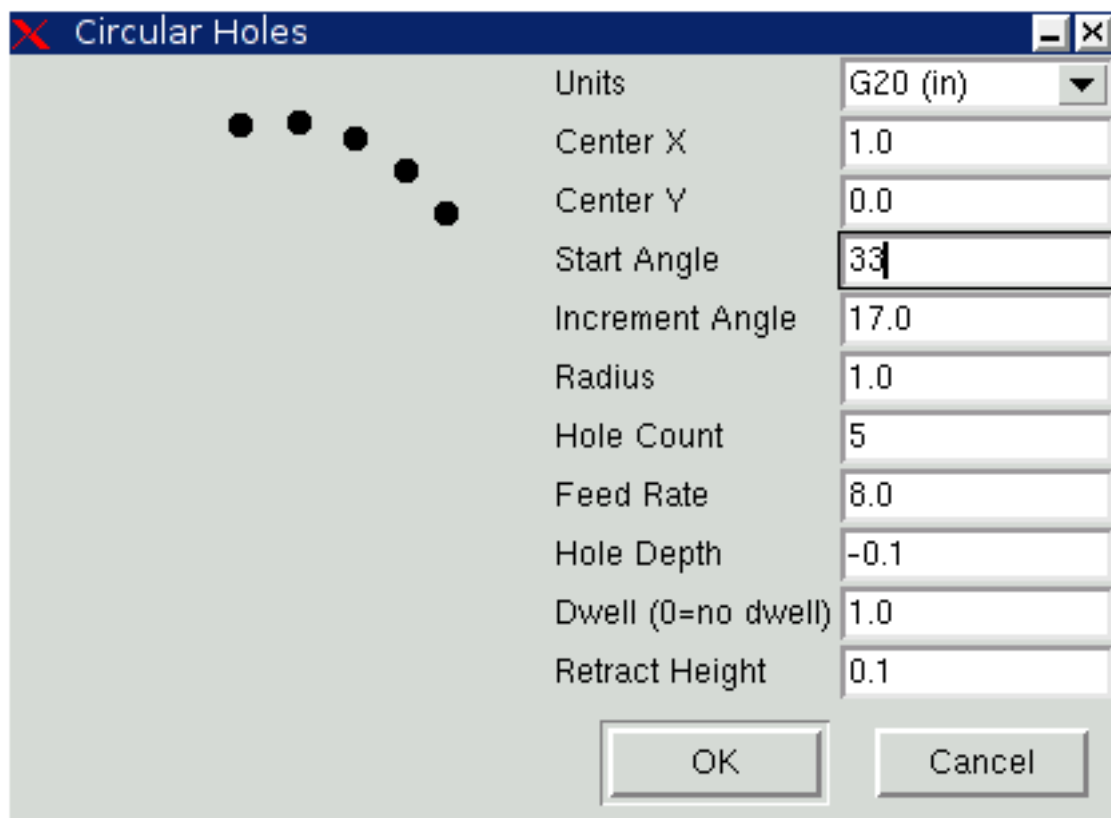


Figure 5.13: Circular Holes

If the environment variable *AXIS_PROGRESS_BAR* is set, then lines written to stderr of the form

```
FILTER_PROGRESS=%d
```

will set the AXIS progress bar to the given percentage. This feature should be used by any filter that runs for a long time.

5.1.12.2 The X Resource Database

The colors of most elements of the AXIS user interface can be customized through the X Resource Database. The sample file *axis_light_background* changes the colors of the backplot window to a *dark lines on white background* scheme, and also serves as a reference for the configurable items in the display area. The sample file *axis_big_dro* changes the position readout to a larger size font. To use these files:

```
xrdb -merge /usr/share/doc/emc2/axis_light_background
xrdb -merge /usr/share/doc/emc2/axis_big_dro
```

For information about the other items which can be configured in Tk applications, see the Tk man pages.

Because modern desktop environments automatically make some settings in the X Resource Database that adversely affect AXIS, by default these settings are ignored. To make the X Resource Database items override AXIS defaults, include the following line in your X Resources:

```
*AXIS*optionLevel: widgetDefault
```

this causes the built-in options to be created at the option level *widgetDefault*, so that X Resources (which are level *userDefault*) can override them.

5.1.12.3 Jogwheel

To improve the interaction of AXIS with a physical jogwheel, the current active axis selected in the GUI is also reported on a *HAL pin* with a name like *axisui.jog.x*. Except for a short time after the current axis has changed, only one of these pins at a time is *TRUE*, the others remain *FALSE*.

After AXIS has created these *HAL pins*, it runs the HAL file declared with: [HAL]POSTGUI_HALFILE. What differs from [HAL]HALFILE, which can only be used once.

5.1.12.4 ~/.axisrc

If it exists, the contents of *~/.axisrc* are executed as Python source code just before the AXIS GUI is displayed. The details of what may be written in the *~/.axisrc* are subject to change during the development cycle.

The following adds Control-Q as a keyboard shortcut for Quit.

Example of .axisrc file

```
root_window.bind("<Control-q>", "destroy .")
help2.append(("Control-Q", "Quit"))
```

The following stops the "Do you really want to quit" dialog.

```
root_window.tk.call("wm","protocol",".", "WM_DELETE_WINDOW","destroy .")
```

5.1.12.5 USER_COMMAND_FILE

A configuration-specific python file may be specified with an INI file setting `[DISPLAY]USER_COMMAND_FILE`. Like a `~/.axisrc` file, this file is sourced just before the AXIS GUI is displayed. This file is specific to an INI file configuration not the user's home directory.

5.1.12.6 user_live_update()

The AXIS GUI includes a no-op (placeholder) function named `user_live_update()` that is executed at the conclusion of the `update()` function of its LivePlotter class. This function may be implemented within a `~/.axisrc` python script or a `[DISPLAY]USER_COMMAND_FILE` python script to make custom, periodic actions. The details of what may be accomplished in this function are dependent on the AXIS GUI implementation and subject to change during the development cycle.

5.1.12.7 user_hal_pins()

The AXIS GUI includes a no-op (placeholder) function named `user_hal_pins()`.

It is executed just after the `.axisrc` file is called and just before any GladeVCP panels / embedded tabs are initialized.

This function may be implemented within a `~/.axisrc` python script or a `[DISPLAY]USER_COMMAND_FILE` python script to make custom HAL pins that use the `axisui.` prefix.

Use `comp` as the HAL component instance reference.

HAL `comp.ready()` is called just after this function returns.

5.1.12.8 External Editor

The menu options `File > Edit...` and `File > Edit Tool Table...` become available after defining the editor in the INI section `[DISPLAY]`. Useful values include `EDITOR=gedit` and `EDITOR=gnome-terminal -e vim`. For more information, see the [Display Section](#) of the INI Configuration Chapter.

5.1.12.9 Virtual Control Panel

AXIS can display a custom virtual control panel in the right-hand pane. You can program buttons, indicators, data displays and more. For more information, see the [PyVCP](#) and the [GladeVCP](#) chapters.

5.1.12.10 Preview Control

Special comments can be inserted into the G-code file to control how the preview of AXIS behaves. In the case where you want to limit the drawing of the preview use these special comments. Anything between the `(AXIS,hide)` and `(AXIS,show)` will not be drawn during the preview. The `(AXIS,hide)` and `(AXIS,show)` must be used in pairs with the `(AXIS,hide)` being first. Anything after a `(AXIS,stop)` will not be drawn during the preview.

These comments are useful to unclutter the preview display (for instance while debugging a larger G-code file, one can disable the preview on certain parts that are already working OK).

- `(AXIS,hide)` Stops the preview (must be first)
- `(AXIS,show)` Resumes the preview (must follow a hide)
- `(AXIS,stop)` Stops the preview from here to the end of the file.
- `(AXIS,notify,the_text)` Displays the `the_text` as an info display

This display can be useful in the AXIS preview when `(debug,message)` comments are not displayed.

5.1.13 Axisui

To improve the interaction of AXIS with physical jog wheels, the axis currently selected in the GUI is also reported on a pin with a name like *axisui.jog.x*. One of these pins is *TRUE* at one time, and the rest are *FALSE*. These are meant to control motion's jog-enable pins.

Axisui Pins AXIS has HAL pins to indicate which jog radio button is selected in the *Manual Control* tab.

Type	Dir	Name
bit	OUT	axisui.jog.x
bit	OUT	axisui.jog.y
bit	OUT	axisui.jog.z
bit	OUT	axisui.jog.a
bit	OUT	axisui.jog.b
bit	OUT	axisui.jog.c
bit	OUT	axisui.jog.u
bit	OUT	axisui.jog.v
bit	OUT	axisui.jog.w

AXIS has a HAL pin to indicate the jog increment selected on the *Manual Tab*.

Type	Dir	Name
float	OUT	axisui.jog.increment

AXIS has a HAL output pin that indicates when an abort has occurred. The *axisui.abort* pin will be *TRUE* and come back to *FALSE* after 0.3ms.

Type	Dir	Name
bit	OUT	axisui.abort

AXIS has a HAL output pin that indicates when an error has occurred. The *axisui.error* pin will remain *TRUE* until all error notifications have been dismissed.

Type	Dir	Name
bit	OUT	axisui.error

AXIS has HAL input pins to clear the pop up notifications for errors and information.

Type	Dir	Name
bit	IN	axisui.notifications-clear
bit	IN	axisui.notifications-clear-error
bit	IN	axisui.notifications-clear-info

AXIS has a HAL input pin that disables/enables the *Pause/Resume* function.

Type	Dir	Name
bit	IN	axisui.resume-inhibit

5.1.14 AXIS Customization Hints

AXIS is a fairly large and difficult-to-penetrate code base, this is helpful To keep the code stable but makes it difficult to customize.

Here we will show code snippets to modify behaviours or visuals of the screen. Keep in mind the internal code of AXIS can change from time to time.

these snippets are not guaranteed to continue to work - they may need adjustment.

5.1.14.1 The update function

There is a function in AXIS named `user_live_update` that is called every time AXIS updates itself. You can use this to update your own functions.

```
# continuous update function
def user_live_update():
    print('i am printed every update...')
```

5.1.14.2 Disable the Close Dialog

```
# disable the do you want to close dialog
root_window.tk.call("wm","protocol",".", "WM_DELETE_WINDOW","destroy .")
```

5.1.14.3 Change the Text Font

```
# change the font

font = 'sans 11'
fname, fsize = font.split()
root_window.tk.call('font','configure','TkDefaultFont','-family',fname,'-size',fsize)

# redo the text in tabs so they resize for the new default font

root_window.tk.call('.pane.top.tabs','itemconfigure','manual','-text',' Manual - F3 ')
root_window.tk.call('.pane.top.tabs','itemconfigure','mdi','-text',' MDI - F5 ')
root_window.tk.call('.pane.top.right','itemconfigure','preview','-text',' Preview ')
root_window.tk.call('.pane.top.right','itemconfigure','numbers','-text',' DR0 ')

# G-code font is independent

root_window.tk.call('.pane.bottom.t.text','configure','-foreground','blue')
#root_window.tk.call('.pane.bottom.t.text','configure','-foreground','blue','-font',font)
#root_window.tk.call('.pane.bottom.t.text','configure','-foreground','blue','-font',font,'- ←
height','12')
```

5.1.14.4 Modify Rapid Rate with Keyboard Shortcuts

```
# use control + ' or 1-0 as keyboard shortcuts for rapidrate and keep ' or 1-0 for feedrate
# also adds text to quick reference in help
```

```
help1.insert(10,("Control+ ',1..9,0", _("Set Rapid Override from 0% to 100%")),)

root_window.bind('<Control-Key-quotelleft>',lambda event: set_rapidrate(0))
root_window.bind('<Control-Key-1>',lambda event: set_rapidrate(10))
root_window.bind('<Control-Key-2>',lambda event: set_rapidrate(20))
root_window.bind('<Control-Key-3>',lambda event: set_rapidrate(30))
root_window.bind('<Control-Key-4>',lambda event: set_rapidrate(40))
root_window.bind('<Control-Key-5>',lambda event: set_rapidrate(50))
root_window.bind('<Control-Key-6>',lambda event: set_rapidrate(60))
root_window.bind('<Control-Key-7>',lambda event: set_rapidrate(70))
root_window.bind('<Control-Key-8>',lambda event: set_rapidrate(80))
root_window.bind('<Control-Key-9>',lambda event: set_rapidrate(90))
root_window.bind('<Control-Key-0>',lambda event: set_rapidrate(100))
```

```

root_window.bind('<Key-QuoteLeft>', lambda event: set_feedrate(0))
root_window.bind('<Key-1>', lambda event: set_feedrate(10))
root_window.bind('<Key-2>', lambda event: set_feedrate(20))
root_window.bind('<Key-3>', lambda event: set_feedrate(30))
root_window.bind('<Key-4>', lambda event: set_feedrate(40))
root_window.bind('<Key-5>', lambda event: set_feedrate(50))
root_window.bind('<Key-6>', lambda event: set_feedrate(60))
root_window.bind('<Key-7>', lambda event: set_feedrate(70))
root_window.bind('<Key-8>', lambda event: set_feedrate(80))
root_window.bind('<Key-9>', lambda event: set_feedrate(90))
root_window.bind('<Key-0>', lambda event: set_feedrate(100))

```

5.1.14.5 Read the INI file

```

# read an INI file item
machine = inifile.find('EMC', 'MACHINE')
print('machine name =', machine)

```

5.1.14.6 Read linuxcnc Status

```

# linuxcnc status can be read from s.
print(s.actual_position)
print(s.paused)

```

5.1.14.7 Change the current view

```

# set the view of the preview
# valid views are view_x view_y view_y2 view_z view_z2 view_p
commands.set_view_z()

```

5.1.14.8 Creating new AXISUI HAL Pins

```

def user_hal_pins():
    comp.newpin('my-new-in-pin', hal.HAL_BIT, hal.HAL_IN)
    comp.ready()

```

5.1.14.9 Creating new HAL Component and Pins

```

# create a component

mycomp = hal.component('my_component')
mycomp.newpin('idle-led', hal.HAL_BIT, hal.HAL_IN)
mycomp.newpin('pause-led', hal.HAL_BIT, hal.HAL_IN)
mycomp.ready()

# connect pins

hal.new_sig('idle-led', hal.HAL_BIT)
hal.connect('halui.program.is-idle', 'idle-led')
hal.connect('my_component.idle-led', 'idle-led')

```



```
# set a pin

hal.set_p('my_component.pause-led','1')

# get a pin 2,8+ branch

value = hal.get_value('halui.program.is-idle')
print('value is a',type(value),'value of',value)
```

5.1.14.10 Switch Tabs with HAL Pins

```
# HAL pins from a GladeVCP panel will not be ready when user_live_update is run
# to read them you need to put them in a try/except block

# the following example assumes 5 HAL buttons in a GladeVCP panel used to switch
# the tabs in the AXIS screen.
# button names are 'manual-tab', 'mdi-tab', 'preview-tab', 'dro-tab', 'user0-tab'
# the user_0 tab if it exists would be the first GladeVCP embedded tab

# for LinuxCNC 2.8+ branch

def user_live_update():
    try:
        if hal.get_value('gladevcp.manual-tab'):
            root_window.tk.call('.pane.top.tabs','raise','manual')
        elif hal.get_value('gladevcp.mdi-tab'):
            root_window.tk.call('.pane.top.tabs','raise','mdi')
        elif hal.get_value('gladevcp.preview-tab'):
            root_window.tk.call('.pane.top.right','raise','preview')
        elif hal.get_value('gladevcp.numbers-tab'):
            root_window.tk.call('.pane.top.right','raise','numbers')
        elif hal.get_value('gladevcp.user0-tab'):
            root_window.tk.call('.pane.top.right','raise','user_0')
    except:
        pass
```

5.1.14.11 Add a GOTO Home button

```
def goto_home(axis):
    if s.interp_state == linuxcnc.INTERP_IDLE:
        home = inifile.find('JOINT_' + str(inifile.find('TRAJ', 'COORDINATES').upper()). ←
            index(axis)), 'HOME')
        mode = s.task_mode
        if s.task_mode != linuxcnc.MODE_MDI:
            c.mode(linuxcnc.MODE_MDI)
            c.mdi('G53 G0 ' + axis + home)

# make a button to home y axis
root_window.tk.call('button','.pane.top.tabs.fmanual.homey','-text','Home Y','-command',' ←
    goto_home Y','-height','2')

# place the button
root_window.tk.call('grid','.pane.top.tabs.fmanual.homey','-column','1','-row','7','- ←
    columnspan','2','-padx','4','-sticky','w')

# any function called from Tcl needs to be added to TclCommands
TclCommands.goto_home = goto_home
commands = TclCommands(root_window)
```

5.1.14.12 Add Button to manual frame

```
# make a new button and put it in the manual frame

root_window.tk.call('button', '.pane.top.tabs.fmanual.mybutton', '-text', 'My Button', '-command', 'mybutton_clicked', '-height', '2')
root_window.tk.call('grid', '.pane.top.tabs.fmanual.mybutton', '-column', '1', '-row', '6', '-columnspan', '2', '-padx', '4', '-sticky', 'w')

# the above send the "mybutton_clicked" command when clicked
# other options are to bind a press or release (or both) commands to the button
# these can be in addition to or instead of the clicked command
# if instead of then delete '-command', 'mybutton_clicked', from the first line

# Button-1 = left mouse button, 2 = right or 3 = middle

root_window.tk.call('bind', '.pane.top.tabs.fmanual.mybutton', '<Button-1>', 'mybutton_pressed')
root_window.tk.call('bind', '.pane.top.tabs.fmanual.mybutton', '<ButtonRelease-1>', 'mybutton_released')

# functions called from the buttons

def mybutton_clicked():
    print('mybutton was clicked')
def mybutton_pressed():
    print('mybutton was pressed')
def mybutton_released():
    print('mybutton was released')

# any function called from Tcl needs to be added to TclCommands

TclCommands.mybutton_clicked = mybutton_clicked
TclCommands.mybutton_pressed = mybutton_pressed
TclCommands.mybutton_released = mybutton_released
commands = TclCommands(root_window)
```

5.1.14.13 Reading Internal Variables

```
# the following variables may be read from the vars instance
```

```
print(vars.machine.get())
print(vars.emcini.get())

active_codes          = StringVar
block_delete          = BooleanVar
brake                  = BooleanVar
coord_type             = IntVar
display_type           = IntVar
dro_large_font         = IntVar
emcini                 = StringVar
exec_state             = IntVar
feedrate               = IntVar
flood                  = BooleanVar
grid_size              = DoubleVar
has_editor             = IntVar
has_ladder             = IntVar
highlight_line         = IntVar
interp_pause          = IntVar
```

```

interp_state      = IntVar
ja_rbutton        = StringVar
jog_aspeed        = DoubleVar
jog_speed         = DoubleVar
kinematics_type   = IntVar
linuxcnc_top_command = StringVar
machine           = StringVar
max_aspeed        = DoubleVar
max_maxvel        = DoubleVar
max_queued_mdi_commands = IntVar
max_speed         = DoubleVar
maxvel_speed      = DoubleVar
mdi_command       = StringVar
metric           = IntVar
mist              = BooleanVar
motion_mode       = IntVar
on_any_limit      = BooleanVar
optional_stop     = BooleanVar
override_limits   = BooleanVar
program_alpha     = IntVar
queued_mdi_commands = IntVar
rapidrate        = IntVar
rotate_mode       = BooleanVar
running_line      = IntVar
show_distance_to_go = IntVar
show_extents      = IntVar
show_live_plot    = IntVar
show_machine_limits = IntVar
show_machine_speed = IntVar
show_program      = IntVar
show_pyvcppanel   = IntVar
show_rapids       = IntVar
show_tool         = IntVar
show_offsets      = IntVar
spindledir        = IntVar
spindlerate       = IntVar
task_mode         = IntVar
task_paused       = IntVar
task_state        = IntVar
taskfile          = StringVar
teleop_mode       = IntVar
tool              = StringVar
touch_off_system  = StringVar
trajcoordinates   = StringVar
tto_gll           = BooleanVar
view_type         = IntVar

```

5.1.14.14 Hide Widgets

```

# hide a widget
# use 'grid' or 'pack' depending on how it was originally placed
root_window.tk.call('grid','forget','pane.top.tabs.fmanual.jogf.zerohome.tooltouch')

```

5.1.14.15 Change a label

```

# change label of a widget
root_window.tk.call('setup_widget_accel','pane.top.tabs.fmanual.mist','Downdraft')

```

```
# make sure it appears (only needed in this case if the mist button was hidden)
root_window.tk.call('grid', '.pane.top.tabs.fmanual.mist', '-column', '1', '-row', '5', '- ←
    colspan', '2', '-padx', '4', '-sticky', 'w')
```

5.1.14.16 Redirect an existing command

```
# hijack an existing command
# originally the mist button calls the mist function
root_window.tk.call('.pane.top.tabs.fmanual.mist', 'configure', '-command', 'hijacked_command' ←
)

# The new function
def hijacked_command():
    print('hijacked mist command')

# add the function to TclCommands
TclCommands.hijacked_command = hijacked_command
commands = TclCommands(root_window)
```

5.1.14.17 Change the DRO color

```
# change dro screen
root_window.tk.call('.pane.top.right.fnumbers.text', 'configure', '-foreground', 'green', '- ←
    background', 'black')
```

5.1.14.18 Change the Toolbar Buttons

```
# change the toolbar buttons

buW = '3'
buH = '2'
boW = '3'

root_window.tk.call('.toolbar.machine_estop', 'configure', '-image', '', '-text', 'ESTOP', '- ←
    width', buW, '-height', buH, '-borderwidth', boW)
root_window.tk.call('.toolbar.machine_power', 'configure', '-image', '', '-text', 'POWER', '- ←
    width', buW, '-height', buH, '-borderwidth', boW)
root_window.tk.call('.toolbar.file_open', 'configure', '-image', '', '-text', 'OPEN', '-width', ←
    buW, '-height', buH, '-borderwidth', boW)
root_window.tk.call('.toolbar.reload', 'configure', '-image', '', '-text', 'RELOAD', '-width', buW ←
    , '-height', buH, '-borderwidth', boW)
root_window.tk.call('.toolbar.program_run', 'configure', '-image', '', '-text', 'RUN', '-width', ←
    buW, '-height', buH, '-borderwidth', boW)
root_window.tk.call('.toolbar.program_step', 'configure', '-image', '', '-text', 'STEP', '-width' ←
    , buW, '-height', buH, '-borderwidth', boW)
root_window.tk.call('.toolbar.program_pause', 'configure', '-image', '', '-text', 'PAUSE', '- ←
    width', buW, '-height', buH, '-borderwidth', boW)
root_window.tk.call('.toolbar.program_stop', 'configure', '-image', '', '-text', 'STOP', '-width' ←
    , buW, '-height', buH, '-borderwidth', boW)
root_window.tk.call('.toolbar.program_blockdelete', 'configure', '-image', '', '-text', 'Skip /' ←
    , '-width', buW, '-height', buH, '-borderwidth', boW)
root_window.tk.call('.toolbar.program_optpause', 'configure', '-image', '', '-text', 'M1', '- ←
    width', buW, '-height', buH, '-borderwidth', boW)
```

```

root_window.tk.call('.toolbar.view_zoomin','configure','-image','','-text','Zoom+','-width' ←
, buW, '-height', buH, '-borderwidth', boW)
root_window.tk.call('.toolbar.view_zoomout','configure','-image','','-text','Zoom-','-width' ←
, buW, '-height', buH, '-borderwidth', boW)
root_window.tk.call('.toolbar.view_z','configure','-image','','-text','Top X','-width', buW, ←
'-height', buH, '-borderwidth', boW)
root_window.tk.call('.toolbar.view_z2','configure','-image','','-text','Top Y','-width', buW ←
, '-height', buH, '-borderwidth', boW)
root_window.tk.call('.toolbar.view_x','configure','-image','','-text','Right','-width', buW, ←
'-height', buH, '-borderwidth', boW)
root_window.tk.call('.toolbar.view_y','configure','-image','','-text','Front','-width', buW, ←
'-height', buH, '-borderwidth', boW)
root_window.tk.call('.toolbar.view_p','configure','-image','','-text','3D','-width', buW, '- ←
height', buH, '-borderwidth', boW)
root_window.tk.call('.toolbar.rotate','configure','-image','','-text','Rotate','-width', buW ←
, '-height', buH, '-borderwidth', boW)
root_window.tk.call('.toolbar.clear_plot','configure','-image','','-text','Clear','-width', ←
buW, '-height', buH, '-borderwidth', boW)

```

5.1.14.19 Change Plotter Colors

In RGBA format, in this order: jog, rapid, feed, arc, toolchange, probe

```

# change plotter colors
try:
    live_plotter.logger.set_colors((255,0,0,255),
                                   (0,255,0,255),
                                   (0,0,255,255),
                                   (255,255,0,255),
                                   (255,255,255,255),
                                   (0,255,255,255))
except Exception as e:
    print(e)

```

5.2 GMOCCAPY

5.2.1 Introduction

GMOCCAPY is a GUI for LinuxCNC, designed to be used with a touch screen, but can also be used on normal screens with a mouse or hardware buttons and MPG wheels, as it presents HAL Pins for the most common needs. Please find more information in the following.

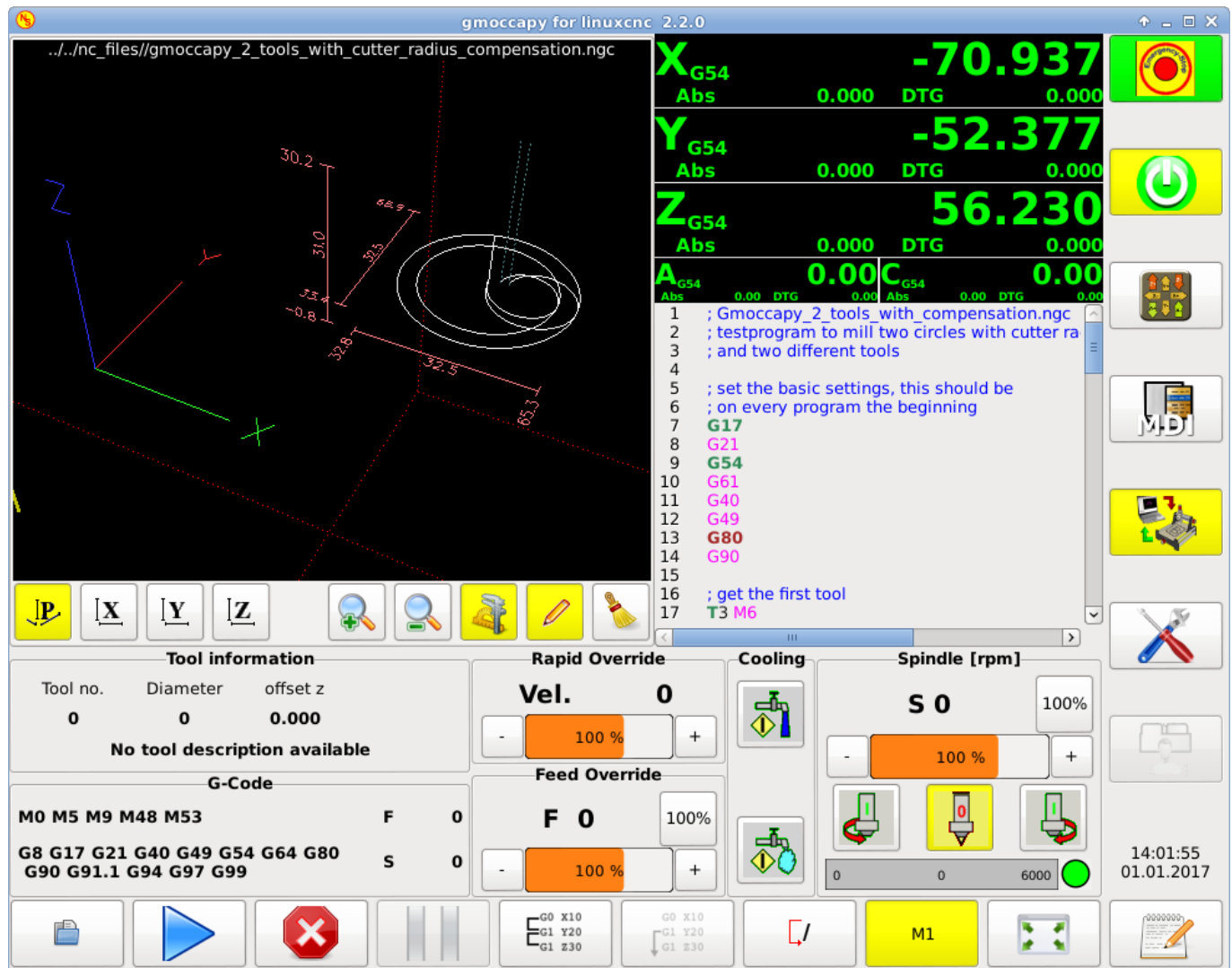
It offers the possibility to display up to 9 axis, support a lathe mode for normal and back tool lathe and can be adapted to nearly every need, because GMOCCAPY supports embedded tabs and side panels. As a good example for that see [gmoccapy_plasma](#).

- GMOCCAPY 3 does support up to 9 axis and 9 joints. As GMOCCAPY 3 has been changed in code to support the joint / axis changes in LinuxCNC it does not work on 2.7 or 2.6 branch!

It has support for integrated virtual keyboard (onboard or matchbox-keyboard), so there is no need for a hardware keyboard or mouse, but it can also be used with that hardware. GMOCCAPY offers a separate settings page to configure most settings of the GUI without editing files.

GMOCCAPY can be localized very easy, because the corresponding files are separated from the linux-cnc.po files, so there is no need to translate unneeded stuff. The files are placed in **/src/po/gmoccapy**.

You could just copy the gmoccapy.pot file to something like it.po and translate that file with gtranslator or poedit. After rebuilding, you'd get the GUI in your preference language. To facilitate the sharing of the translation GMOCCAPY is presented on the [Weblate web interface](#). GMOCCAPY is currently available in English, German, Spanish, Polish, Serbian and Hungarian. Feel free to help me to introduce more languages, be it locally or via the web. If you need help, don't hesitate to contact me on nieson@web.de.



5.2.2 Requirements

GMOCAPY 3 has been tested on Debian Jessie, Debian Stretch and MINT 18 with LinuxCNC master and 2.8 release. It fully support joint / axis changes of LinuxCNC, making it suitable as GUI for Scara, Robots or any other config with more joints than axis. So it do support also gantry configs. if you use other versions, please inform about problems and / or solutions on the [LinuxCNC forum](#) or the [German CNC Ecke Forum](#) or [LinuxCNC users mailing list](#).

The minimum screen resolution for GMOCCAPY, using it without side panels is **979 x 750 Pixel**, so it should fit to every standard screen. It is recommended to use screens with minimum resolution of 1024x748.

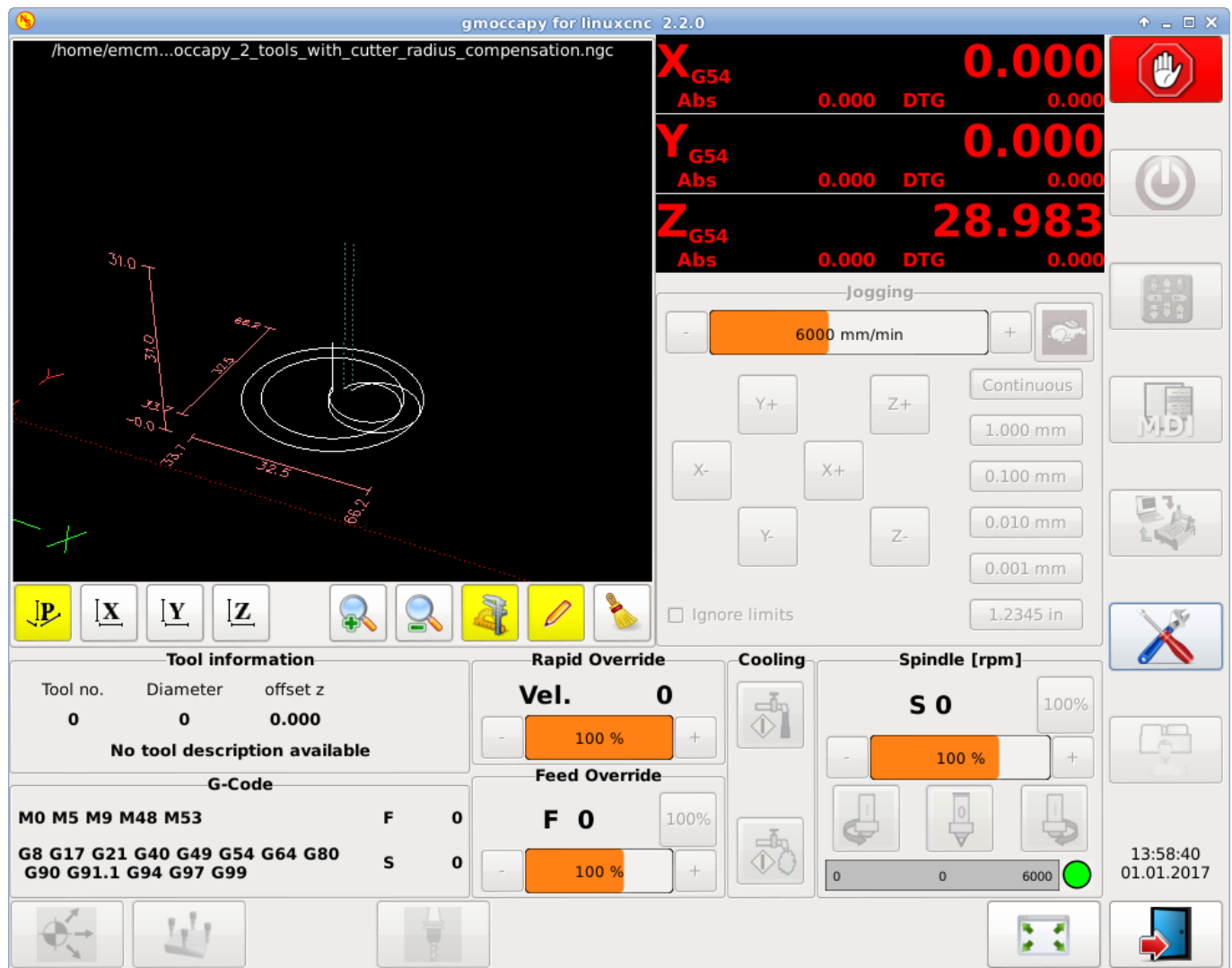
5.2.3 How To Get GMOCCAPY

Beginning with LinuxCNC 2.8 GMOCCAPY 3 is included in the standard installation. So the easiest way to get GMOCCAPY on your controlling PC, is just to get the [ISO](#) and install from the CD / DVD / USB-Stick.

You will receive updates with the regular Debian packages.

GMOCCAPY 3 is only included in the actual 2.8 and master release.

You will get a similar screen to the following: The design may vary depending on your config.



5.2.4 Basic Configuration

There is really not too much to configure just to run GMOCCAPY, but there are some points you should take care of if you want to use all the features of the GUI.

You will find a lot of simulation configurations (INI files) included, just to show the basics:

- gmoccapy.ini
- gmoccapy_4_axis.ini

- lathe_configs/gmoccapy_lathe.ini
- lathe_configs/gmoccapy_lathe_imperial.ini
- gmoccapy_left_panel.ini
- gmoccapy_right_panel.ini
- gmoccapy_messages.ini
- gmoccapy_pendant.ini
- gmoccapy_sim_hardware_button.ini
- gmoccapy_tool_sensor.ini
- gmoccapy_with_user_tabs.ini
- gmoccapy_XYZAB.ini
- gmoccapy_XYZAC.ini
- gmoccapy_XYZCW.ini
- gmoccapy-JA/Gantry/gantry_mm.ini
- gmoccapy-JA/scara/scara.ini
- gmoccapy-JA/table-rotary-tilting/xyzac-trt.ini
- and a lot more ...

The names should explain the main intention of the different INI files.

If you use an existing configuration of your machine, just edit your INI according to this document.



Important

If you want to use [MACROS](#), don't forget to set the path to your macros or subroutines folder as described below.

So let us take a closer look to the the INI file and what you need to include to use GMOCCAPY on your machine.

5.2.4.1 The DISPLAY Section

```
[DISPLAY]
DISPLAY = gmoccapy
REFERENCE_FILE_PATH = gmoccapy_preferences
MAX_FEED_OVERRIDE = 1.5
MAX_SPINDLE_OVERRIDE = 1.2
MIN_SPINDLE_OVERRIDE = 0.5
LATHE = 1
BACK_TOOL_LATHE = 1
PROGRAM_PREFIX = ../../nc_files/
```

The most important part is to tell LinuxCNC to use GMOCCAPY, editing the [DISPLAY] section.

```
[DISPLAY]
DISPLAY = gmoccapy

REFERENCE_FILE_PATH = GMOCCAPY_preferences
```


GMOCCAPY 3 does support the following command line options:

- `-user_mode` : If set, the setup button will be disabled, so normal machine operators are not able to edit the settings of the machine.
- `-logo <path to logo file>` : If given, the logo will hide the jog button tab in manual mode, this is only useful for machines with hardware button for jogging and increment selection.

The line `PREFERENCE_FILE_PATH` gives the location and name of the preferences file to be used. In most cases this line will not be needed, it is used by GMOCCAPY to store your settings of the GUI, like themes, DRO units, colors, and keyboard settings, etc., see [settings page](#) for more details.

Note

If no path or file is given, GMOCCAPY will use as default `<your_machinename>.pref`, if no machine name is given in your INI file it will use `gmoccapy.pref` The file will be stored in your config directory, so the settings will not be mixed if you use several configs. If you only want to use one file for several machines, you need to include `PREFERENCE_FILE_PATH` in your INI.

```
MAX_FEED_OVERRIDE = 1.5
```

Sets the maximum feed override, in the example given, you will be allowed to override the feed by 150%.

Note

If no value is given, it will be set to 1.0

```
MAX_SPINDLE_OVERRIDE = 1.2  
MIN_SPINDLE_OVERRIDE = 0.5
```

Will allow you to change the spindle override within a limit from 50% to 120%.

Note

If no values are given, MAX will be set to 1.0 and MIN to 0.1.

```
LATHE = 1  
BACK_TOOL_LATHE = 1
```

The first line set the screen layout to control a lathe.

The second line is optional and will switch the X axis in a way you need for a back tool lathe. Also the keyboard shortcuts will react in a different way. It is allowed with GMOCCAPY to configure a lathe also with additional axis, so you may use also a XZCW config for a lathe.

Tip

See also the [Lathe Specific Section](#).

- `PROGRAM_PREFIX = ../../nc_files/`

Is the entry to tell LinuxCNC/GMOCCAPY where to look for the ngc files.

Note

If not specified GMOCCAPY will look in the following order for ngc files: linuxcnc/nc_files and then the users home directory.

You can add embedded programs to GMOCCAPY like you can do in AXIS, Touchy and gscreen. All is done by GMOCCAPY automatically if you include a few lines in your INI file in the DISPLAY section.

If you never used a glade panel, I recommend to read the excellent documentation: [GladeVCP](#).

Example

```
EMBED_TAB_NAME = DRO
EMBED_TAB_LOCATION = ntb_user_tabs
EMBED_TAB_COMMAND = gladevcp -x {XID} dro.glade

EMBED_TAB_NAME = Second user tab
EMBED_TAB_LOCATION = ntb_preview
EMBED_TAB_COMMAND = gladevcp -x {XID} vcp_box.glade
```

All you have to take care of, is that you include for every tab or side panel the mentioned three lines:

- EMBED_TAB_NAME = Represents the name of the tab or side panel, it is up to you what name you use, but it must be present!
- EMBED_TAB_LOCATION = The place where your program will be placed in the GUI, see figure [Embedded tab locations](#). Valid values are:
 - **ntb_user_tabs** (as main tab, covering the complete screen)
 - **ntb_preview** (as tab on the preview side **(1)**)
 - **hbox_jog** (will hide the jog buttons and introduce your glade file here **(2)**)
 - **box_left** (on the left, complete high of the screen)
 - **box_right** (on the right, in between the normal screen and the button list)
 - **box_tool_and_code_info** (will hide the Tool information and G-code frames and introduce your glade file here **(3)**)
 - **box_tool_info** (will hide the Tool information frame and introduce your glade file here)
 - **box_code_info** (will hide the G-code information frame and introduce your glade file here)
 - **box_vel_info** (will hide the velocity frames and introduce your glade file **(4)**)
 - **box_coolant_and_spindle** (will hide the coolant and spindle frames and introduce your glade file here **(5)+(6)**)
 - **box_cooling** (will hide the cooling frame and introduce your glade file **(5)**)
 - **box_spindle** (will hide the spindle frame and introduce your glade file **(6)**)
 - **box_custom_1** (will introduce your glade file left of vel_frame)
 - **box_custom_2** (will introduce your glade file left of cooling_frame)
 - **box_custom_3** (will introduce your glade file left of spindle_frame)
 - **box_custom_4** (will introduce your glade file right of spindle_frame)

Note

See also the included sample INI files to see the differences.

- EMBED_TAB_COMMAND = The command to execute, i.e.
-

```
gladevcp -x {XID} dro.glade
```

includes a custom glade file called dro.glade in the mentioned location The file must be placed in the config folder of your machine.

```
gladevcp h_buttonlist.glade
```

will just open a new user window called h_buttonlist.glade note the difference, this one is stand alone, and can be moved around independent from GMOCCAPY window.

```
gladevcp -c gladevcp -u hitcounter.py -H manual-example.hal manual-example.ui
```

will add a the panel manual-example.ui, include a custom python handler, hitcounter.py and make all connections after realizing the panel according to manual-example.hal.

```
hide
```

will hide the chosen box.

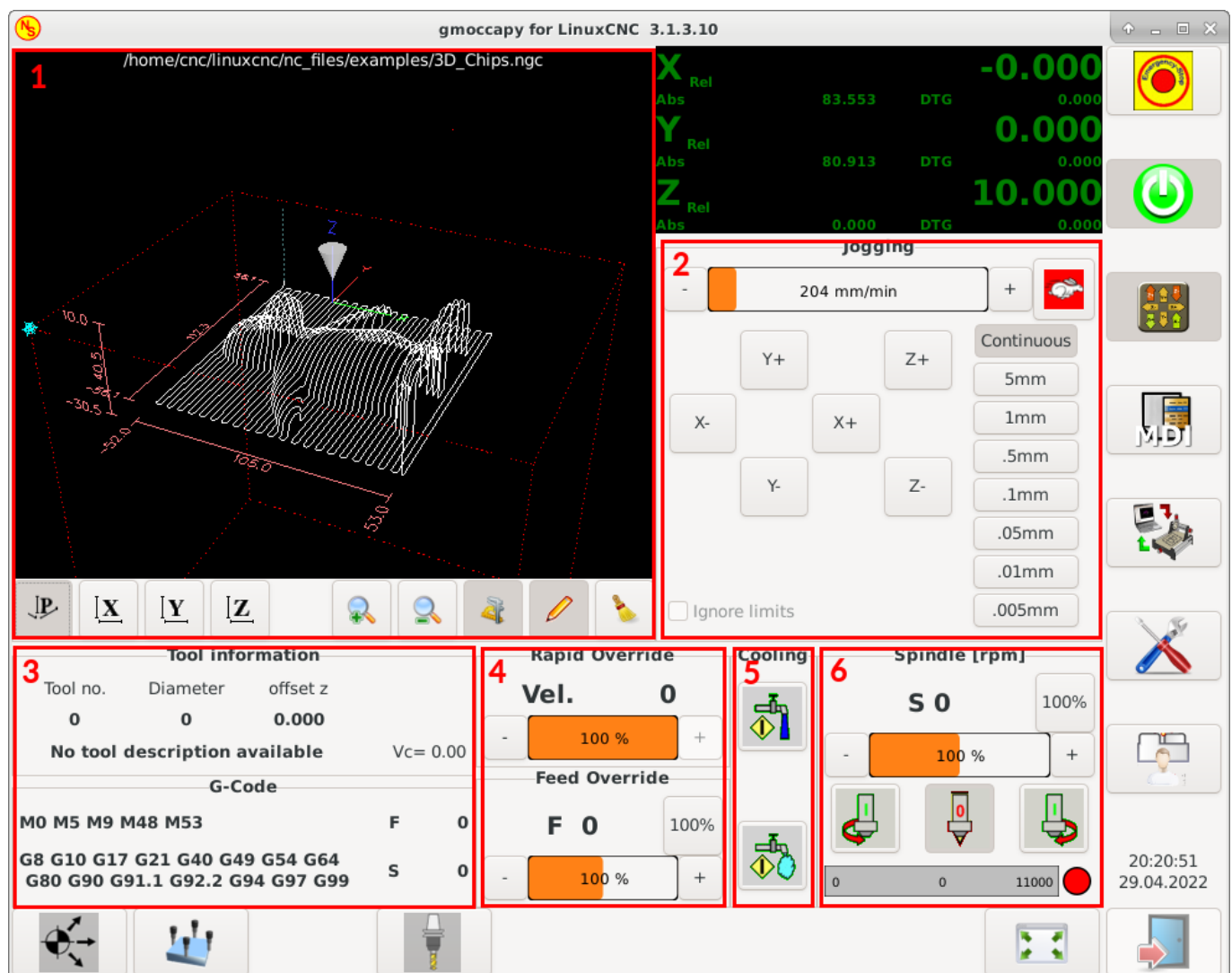


Figure 5.14: Embedded tab locations

Note

If you make any HAL connections to your custom glade panel, you need to do that in the HAL file specified in the `EMBED_TAB_COMMAND` line, otherwise you may get an error that the HAL pin does not exist — this is because of race conditions loading the HAL files. Connections to `GMOCCAPY` HAL pins need to be made in the postgui HAL file specified in your INI file, because these pins do not exist prior of realizing the GUI.

Here are some examples:



Figure 5.15: ntb_preview - in maximized view

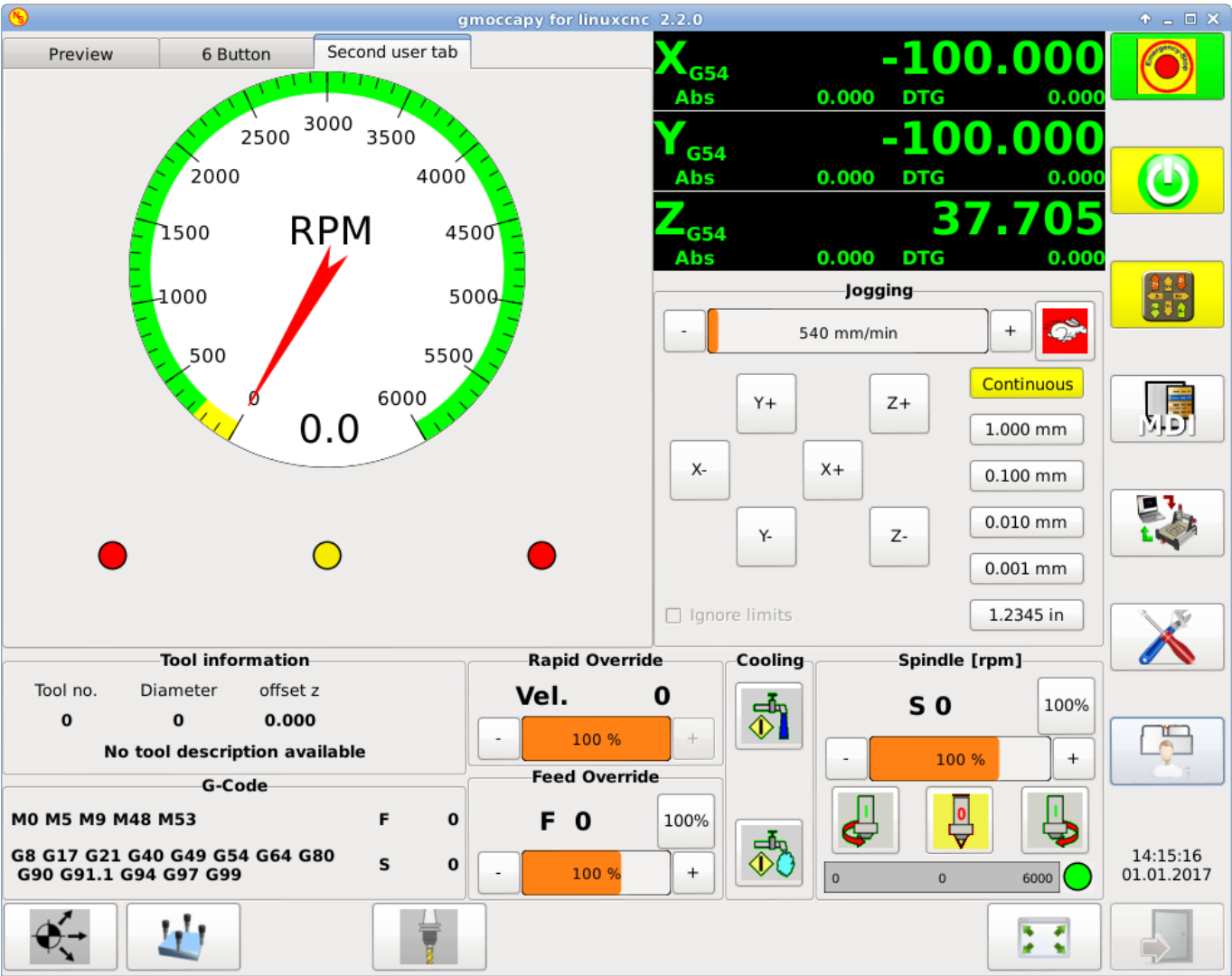
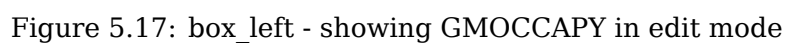


Figure 5.16: ntb_preview



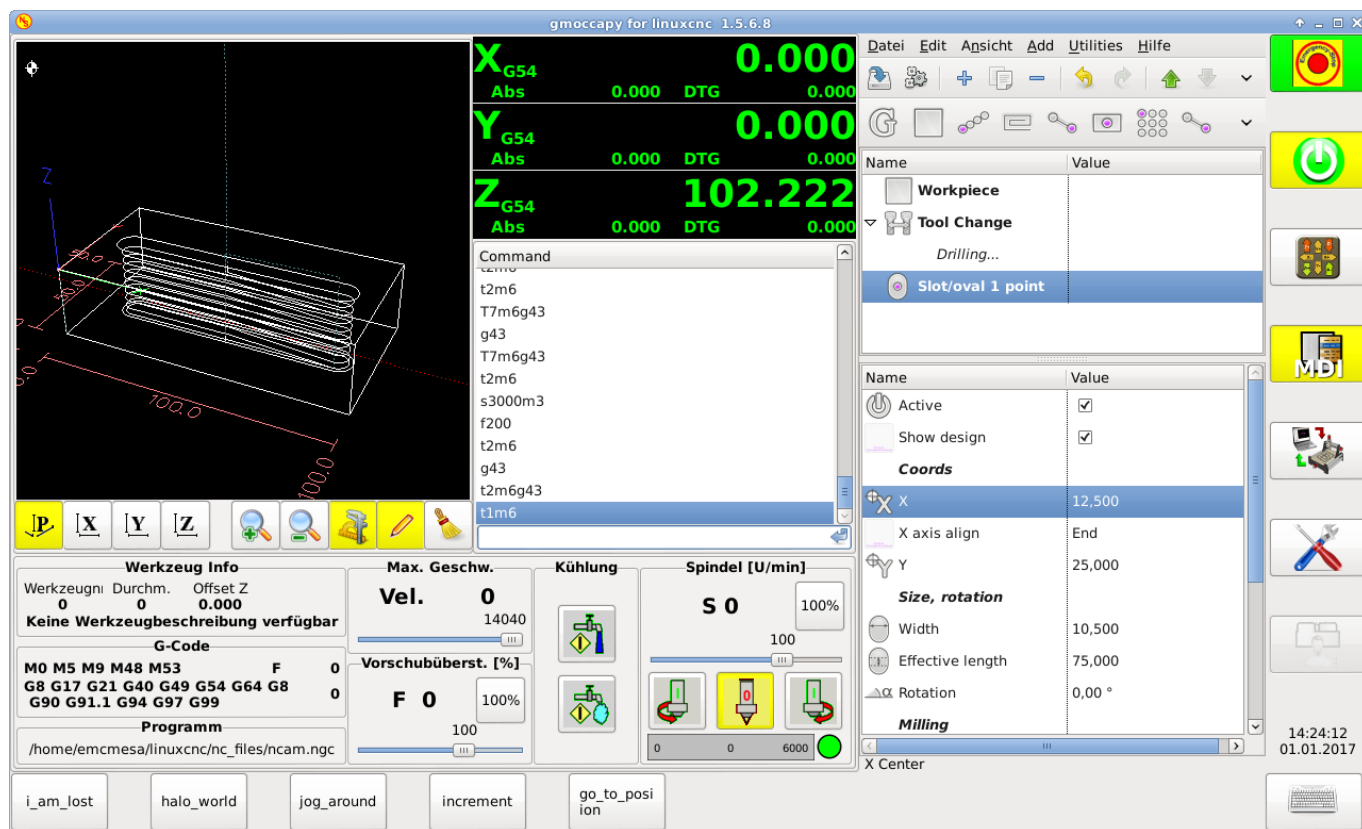


Figure 5.18: box_right - and GMOCCAPY in MDI mode

GMOCCAPY has the ability to create HAL driven user messages. To use them you need to introduce some lines in the [DISPLAY] section of the INI file.

Here is how to set up 3 user pop up message dialogs the messages support pango markup language. Detailed information about the markup language can be found at [Pango Markup](#).

```
MESSAGE_TEXT    = The text to be displayed, may be pango markup formatted
MESSAGE_TYPE    = "status" , "okdialog" , "yesnodelist"
MESSAGE_PINNAME = is the name of the HAL pin group to be created
```

- *status* : Will just display a message as pop up window, using the messaging system of GMOCCAPY
- *okdialog* : Will hold focus on the message dialog and will activate a "-waiting" Hal_Pin OUT. Closing the message will reset the waiting pin
- *yesnodelist* : Will hold focus on the message dialog and will activate a "-waiting" Hal_Pin bit OUT it will also give access to an "-response" Hal_Pin Bit Out, this pin will hold 1 if the user clicks OK, and in all other states it will be 0 Closing the message will reset the waiting pin The response HAL Pin will remain 1 until the dialog is called again

Example

```
MESSAGE_TEXT = This is a <span background="#ff0000" foreground="#ffffff">
info-message</span> test
MESSAGE_TYPE = status
MESSAGE_PINNAME = statustest

MESSAGE_TEXT = This is a yes no dialog test
```

```
MESSAGE_TYPE = yesnodialog
MESSAGE_PINNAME = yesnodialog

MESSAGE_TEXT = Text can be <small>small</small>, <big>big</big>, <b>bold</b> <i>italic</i>, ↵
    and even be <span color="red">colored</span>.
MESSAGE_TYPE = okdialog
MESSAGE_PINNAME = okdialog
```

The specific HAL pin conventions for these can be found under the [User Messages](#) HAL pin section.

5.2.4.2 The RS274NGC Section

```
[RS274NGC]
SUBROUTINE_PATH = macros
```

Sets the path to search for macros and other subroutines. If you want to use several subroutine paths, just separate them with ":".

5.2.4.3 The MACRO Section

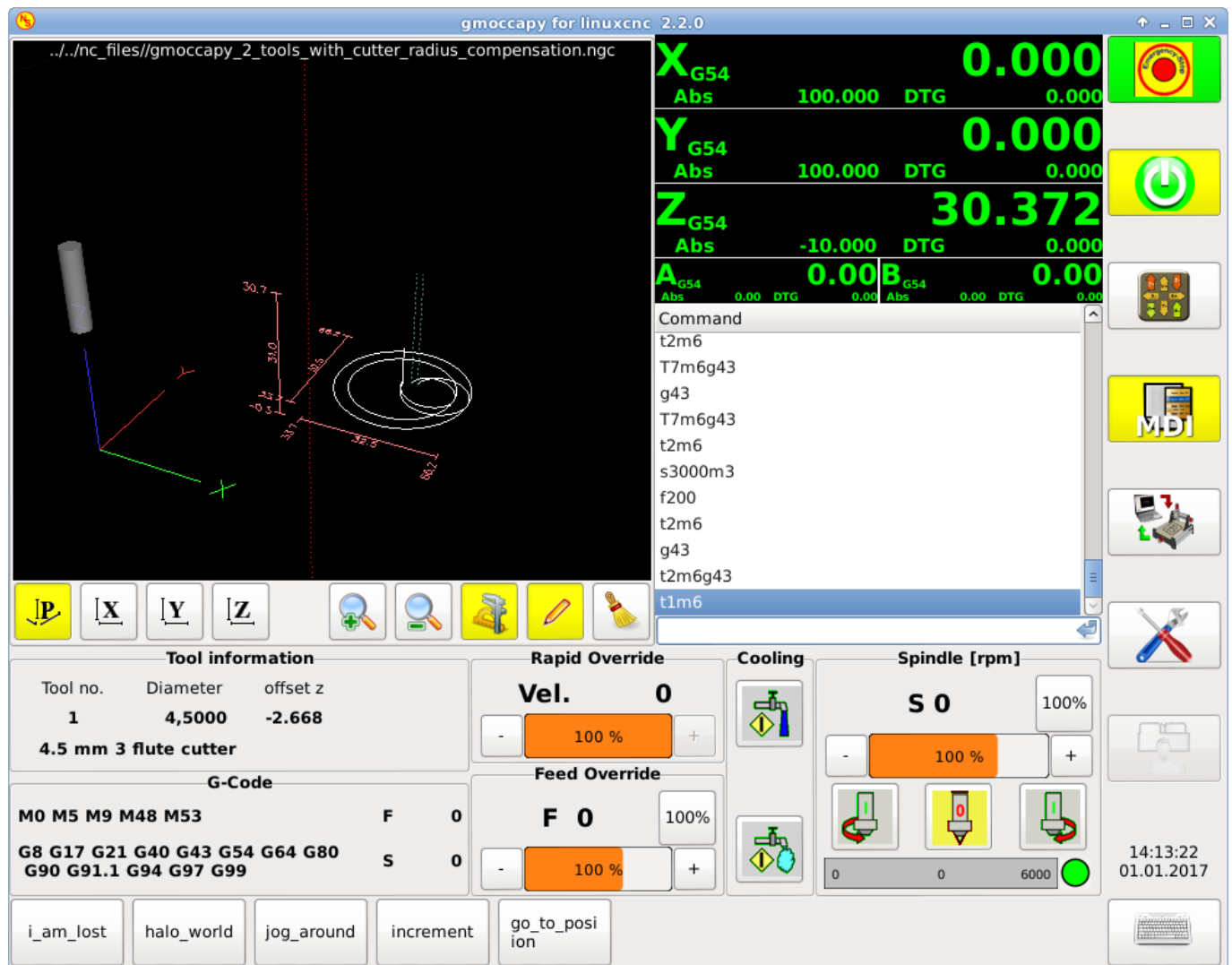
You can add macros to GMOCCAPY, similar to Touchy's way. A macro is nothing else than a ngc-file. You are able to execute complete CNC programs in MDI mode, by just pushing one button. To do so, you have to add a section like so:

```
[MACROS]
MACRO = i_am_lost
MACRO = hello_world
MACRO = jog_around
MACRO = increment xinc yinc
MACRO = go_to_position X-pos Y-pos Z-pos
```

This will add 5 macros to the MDI button list.

Note

As maximum 16 macros will appear in the GUI, due to space reasons you may need to click on an arrow to switch page and display hidden macro button. It is no error placing more in your INI file. The macro button will be displayed in the order of the INI entries.



The name of the file must be **exactly the same** as the name given in the MACRO line. So the macro **i_am_lost** will call the file **i_am_lost.ngc**.

The macro ngc files must follow some rules:

- the name of the file need to be exactly the same as the name mentioned in the macro line, just with the ngc extension (case sensitive)
- The file must contain a subroutine like so: **O<i_am_lost> sub**, the name of the sub must match exactly (**case sensitive**) the name of the macro
- the file must end with an endsub **O<i_am_lost> endsub** followed by an **M2** command
- the files need to be placed in a folder specified in your INI file in the [RS274NGC](#) section.

The code in between sub and endsub will be executed by pushing the corresponding macro button.

Note

You will find the sample macros in macros folder placed in the GMOCCAPY sim folder. If you have given several subroutine paths, they will be searched in the order of the given paths. The first file found will be used.

GMOCCAPY will also accept macros asking for parameters like:

```
go_to_position X-pos Y-pos Z-pos
```

The parameters must be separated by spaces. This calls a file *go_to_position.ngc* with the following content:

```
; Test file go to position
; will jog the machine to a given position

O<go_to_position> sub

G17
G21
G54
G61
G40
G49
G80
G90

;#1 = <X-Pos>
;#2 = <Y-Pos>
;#3 = <Z-Pos>

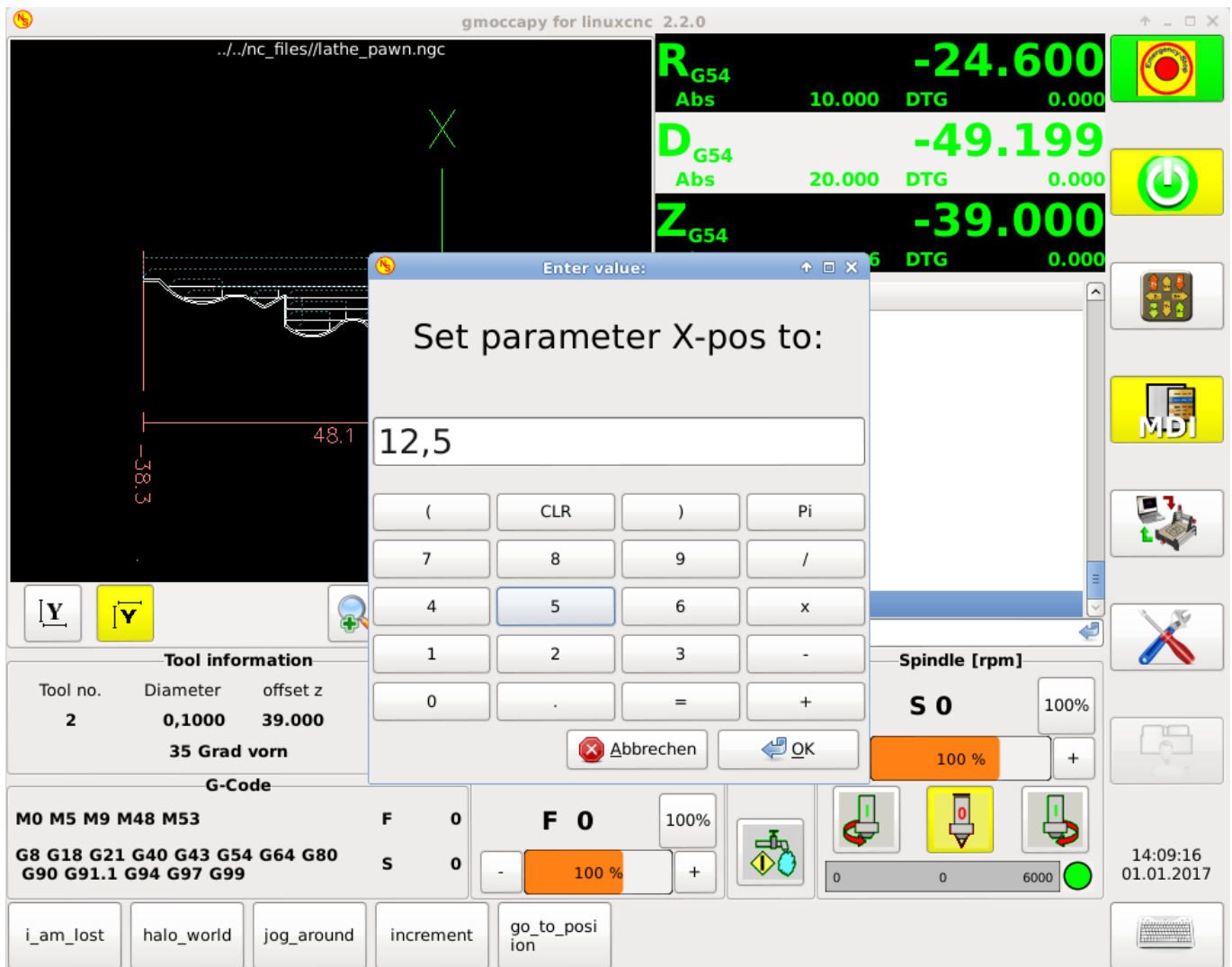
(DBG, Will now move machine to X = #1 , Y = #2 , Z = #3)
G0 X #1 Y #2 Z #3

O<go_to_position> endsub
M2
```

After pushing the **execute macro button**, you will be asked to enter the values for **X-pos Y-pos Z-pos** and the macro will only run if all values have been given.

Note

If you would like to use a macro without any movement, see also the notes in [known problems](#).



5.2.4.4 The TRAJ Section

```
DEFAULT_LINEAR_VELOCITY = 85.0
MAX_VELOCITY = 230.000
```

Sets the maximal velocity and the default jog velocity of the machine.

Note

If `DEFAULT_LINEAR_VELOCITY` is not given, half of `MAX_VELOCITY` will be used. If that value is also not given, it will default to 180. If `MAX_VELOCITY` is not given, it will default to 600.

5.2.5 HAL Pins

GMOCCAPY exports several HAL pins to be able to react to hardware devices. The goal is to get a GUI that may be operated in a tool shop, completely/mostly without mouse or keyboard.

Note

You will have to do all connections to GMOCCAPY pins in your postgui.hal file. When GMOCCAPY is started it creates the HAL pins for the GUI then it executes the HAL file named in the INI file: `[HAL]POSTGUI_HALFILE=<filename>`.

Typically `<filename>` would be the configs base name + `_postgui` + `.hal` eg. `lathe_postgui.hal`, but can be any legal filename.

These commands are executed after the screen is built, guaranteeing the widget's HAL pins are available.

You can have multiple line of `POSTGUI_HALFILE=<filename>` in the INI.

Each will be run one after the other in the order they appear.

See [Tabs and side panels](#) for details.

5.2.5.1 Right And Bottom Button Lists

The screen has two main button lists, one on the right side an one on the bottom. The right handed buttons will not change during operation, but the bottom button list will change very often. The buttons are count from up to down and from left to right beginning with "0".

Note

the pin name for **GMOCCAPY2** has changed to order them better:

In `hal_show` you will see the right (vertical) buttons are:

- `gmoccap.v-button.button-0`
- `gmoccap.v-button.button-1`
- `gmoccap.v-button.button-2`
- `gmoccap.v-button.button-3`
- `gmoccap.v-button.button-4`
- `gmoccap.v-button.button-5`
- `gmoccap.v-button.button-6`

and the bottom (horizontal) buttons are:

- `gmoccap.h-button.button-0`
 - `gmoccap.h-button.button-1`
 - `gmoccap.h-button.button-2`
 - `gmoccap.h-button.button-3`
 - `gmoccap.h-button.button-4`
 - `gmoccap.h-button.button-5`
 - `gmoccap.h-button.button-6`
 - `gmoccap.h-button.button-7`
 - `gmoccap.h-button.button-8`
-

- gmoccapy.h-button.button-9

As the buttons in the bottom list will change according the mode and other influences, the hardware buttons will activate different functions, and you don't have to take care about switching functions around in HAL, because that is done completely by GMOCCAPY!

For a 3 axis XYZ machine the HAL pin will react as follows.

In manual mode:

- gmoccapy.h-button.button-0 == open homing button
- gmoccapy.h-button.button-1 == open touch off stuff
- gmoccapy.h-button.button-2 ==
- gmoccapy.h-button.button-3 == open tool dialogs
- gmoccapy.h-button.button-4 ==
- gmoccapy.h-button.button-5 ==
- gmoccapy.h-button.button-6 ==
- gmoccapy.h-button.button-7 ==
- gmoccapy.h-button.button-8 == full-size preview
- gmoccapy.h-button.button-9 == exit if machine is off, otherwise no reaction

In mdi mode:

- gmoccapy.h-button.button-0 == macro_0 or nothing
- gmoccapy.h-button.button-1 == macro_1 or nothing
- gmoccapy.h-button.button-2 == macro_2 or nothing
- gmoccapy.h-button.button-3 == macro_3 or nothing
- gmoccapy.h-button.button-4 == macro_4 or nothing
- gmoccapy.h-button.button-5 == macro_5 or nothing
- gmoccapy.h-button.button-6 == macro_6 or nothing
- gmoccapy.h-button.button-7 == macro_7 or nothing
- gmoccapy.h-button.button-8 == macro_8 or switch page to additional macros
- gmoccapy.h-button.button-9 == open keyboard or abort if macro is running

In auto mode:

- gmoccapy.h-button.button-0 == open file
 - gmoccapy.h-button.button-1 == reload program
 - gmoccapy.h-button.button-2 == run
 - gmoccapy.h-button.button-3 == stop
 - gmoccapy.h-button.button-4 == pause
 - gmoccapy.h-button.button-5 == step by step
-

- gmoccap.h-button.button-6 == run from line if enabled in settings, otherwise Nothing
- gmoccap.h-button.button-7 == optional blocks
- gmoccap.h-button.button-8 == full-size preview
- gmoccap.h-button.button-9 == edit code

In settings mode:

- gmoccap.h-button.button-0 == delete MDI history
- gmoccap.h-button.button-1 ==
- gmoccap.h-button.button-2 ==
- gmoccap.h-button.button-3 ==
- gmoccap.h-button.button-4 == open classic ladder
- gmoccap.h-button.button-5 == open HAL scope
- gmoccap.h-button.button-6 == open HAL status
- gmoccap.h-button.button-7 == open HAL meter
- gmoccap.h-button.button-8 == open HAL calibration
- gmoccap.h-button.button-9 == open HAL show

In homing mode:

- gmoccap.h-button.button-0 ==
- gmoccap.h-button.button-1 == home all
- gmoccap.h-button.button-2 ==
- gmoccap.h-button.button-3 == home x
- gmoccap.h-button.button-4 == home y
- gmoccap.h-button.button-5 == home z
- gmoccap.h-button.button-6 ==
- gmoccap.h-button.button-7 ==
- gmoccap.h-button.button-8 == unhome all
- gmoccap.h-button.button-9 == back

In touch off mode:

- gmoccap.h-button.button-0 == edit offsets
 - gmoccap.h-button.button-1 == touch X
 - gmoccap.h-button.button-2 == touch Y
 - gmoccap.h-button.button-3 == touch Z
 - gmoccap.h-button.button-4 ==
 - gmoccap.h-button.button-5 ==
-

- gmoccap.h-button.button-6 == zero G92
- gmoccap.h-button.button-7 ==
- gmoccap.h-button.button-8 == set selected
- gmoccap.h-button.button-9 == back

In tool mode:

- gmoccap.h-button.button-0 == delete tool(s)
- gmoccap.h-button.button-1 == new tool
- gmoccap.h-button.button-2 == reload tool table
- gmoccap.h-button.button-3 == apply changes
- gmoccap.h-button.button-4 == change tool by number T? M6
- gmoccap.h-button.button-5 == set tool by number without change M61 Q?
- gmoccap.h-button.button-6 == change tool to the selected one
- gmoccap.h-button.button-7 ==
- gmoccap.h-button.button-8 == touch of tool in Z
- gmoccap.h-button.button-9 == back

In edit mode:

- gmoccap.h-button.button-0 ==
- gmoccap.h-button.button-1 == reload file
- gmoccap.h-button.button-2 == save
- gmoccap.h-button.button-3 == save as
- gmoccap.h-button.button-4 ==
- gmoccap.h-button.button-5 ==
- gmoccap.h-button.button-6 == new file
- gmoccap.h-button.button-7 ==
- gmoccap.h-button.button-8 == show keyboard
- gmoccap.h-button.button-9 == back

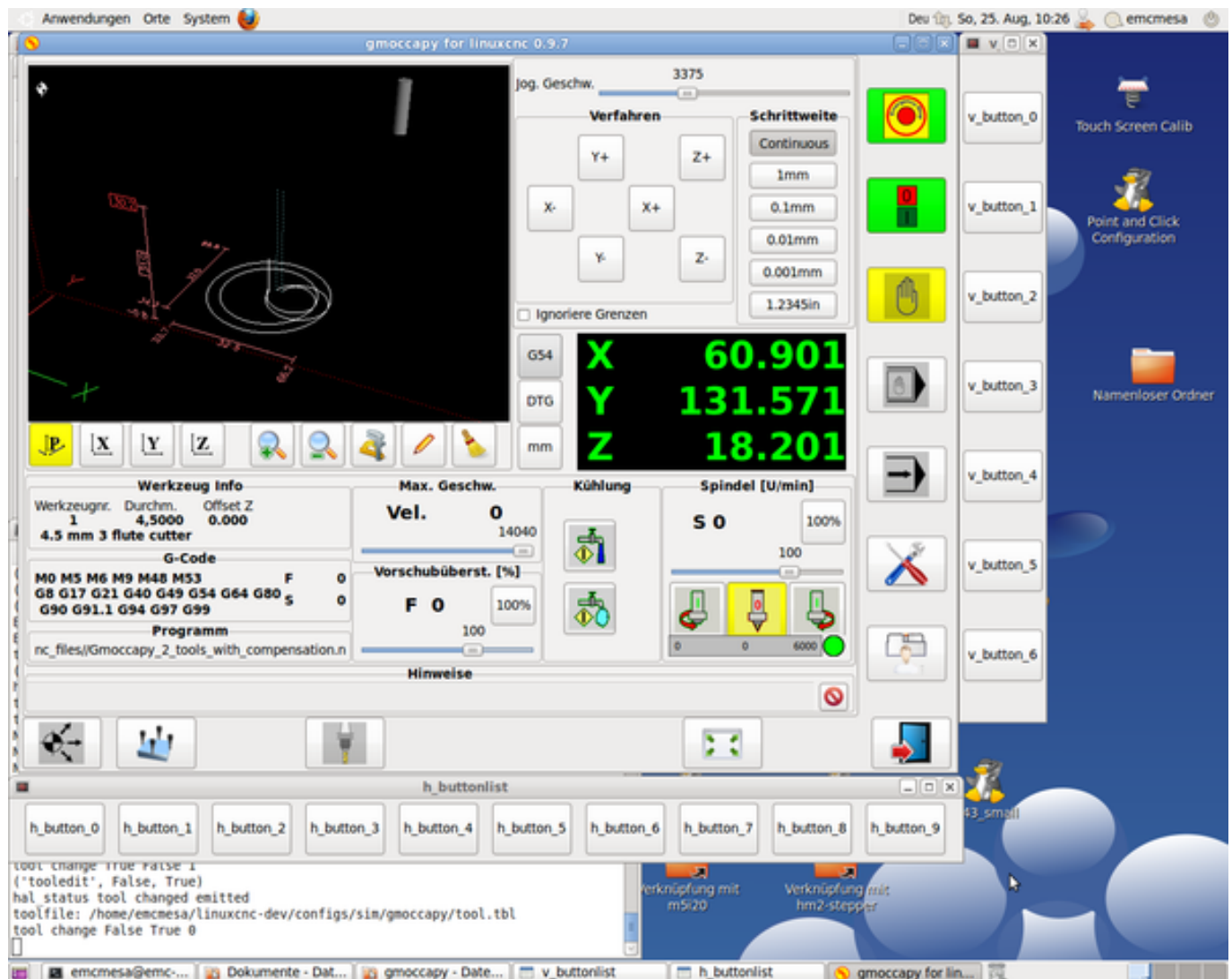
In select file mode:

- gmoccap.h-button.button-0 == go to home directory
 - gmoccap.h-button.button-1 == one directory level up
 - gmoccap.h-button.button-2 ==
 - gmoccap.h-button.button-3 == move selection left
 - gmoccap.h-button.button-4 == move selection right
 - gmoccap.h-button.button-5 == jump to directory as set in settings
-

- gmoccapy.h-button.button-6 ==
- gmoccapy.h-button.button-7 == select / ENTER
- gmoccapy.h-button.button-8 ==
- gmoccapy.h-button.button-9 == back

So we have 67 reactions with only 10 HAL pin!

These pins are made available to be able to use the screen without an touch panel, or protect it from excessive use by placing hardware buttons around the panel.



5.2.5.2 Velocities And Overrides

All sliders from GMOCCAPY can be connected to hardware encoder or hardware potentiometers.

Note

for GMOCCAPY 3 the HAL pin name has changed, as new controls has been implemented, max velocity does not exist any more, as rapid override has been implemented. This change has been done as many user ask for that.

To connect *encoders* the following pin are exported:

PIN	TYPE	FUNCTION
gmoccapy.jog.jog-velocity.counts	HAL_S32	Jog velocity
gmoccapy.jog.jog-velocity.count-enable	HAL_BIT	Must be True, to enable counts
gmoccapy.feed.feed-override.counts	HAL_S32	feed override
gmoccapy.feed.feed-override.count-enable	HAL_BIT	Must be True, to enable counts
gmoccapy.feed.reset-feed-override	HAL_BIT	reset the feed override to 100%
gmoccapy.spindle.spindle-override.counts	HAL_S32	spindle override
gmoccapy.spindle.spindle-override.count-enable	HAL_BIT	Must be True, to enable counts
gmoccapy.spindle.reset-spindle-override	HAL_BIT	reset the spindle override to 100%
gmoccapy.rapid.rapid-override.counts	HAL_S32	Maximal Velocity of the machine
gmoccapy.rapid.rapid-override.count-enable	HAL_BIT	Must be True, to enable counts

To connect *potentiometers*, use the following HAL pin:

PIN	TYPE	FUNCTION
gmoccapy.jog.jog-velocity.direct-value	HAL_FLOAT	To adjust the jog velocity slider
gmoccapy.jog.jog-velocity.analog-enable	HAL_BIT	Must be True, to allow analog inputs
gmoccapy.feed.feed-override.direct-value	HAL_FLOAT	To adjust the feed override slider
gmoccapy.feed.feed-override.analog-enable	HAL_BIT	Must be True, to allow analog inputs
gmoccapy.spindle.spindle-override.direct-value	HAL_FLOAT	To adjust the spindle override slider
gmoccapy.spindle.spindle-override.analog-enable	HAL_BIT	Must be True, to allow analog inputs
gmoccapy.rapid.rapid-override.direct-value	HAL_FLOAT	To adjust the max velocity slider
gmoccapy.rapid.rapid-override.analog-enable	HAL_BIT	Must be True, to allow analog inputs

In addition GMOCCAPY 3 offers additional HAL pin to control the new slider widgets with momentary switches. The values how fast the increase or decrease will be, must be set in the glade file. In a future release it will be integrated in the settings page.

PIN	TYPE	FUNCTION
SPEED		
gmoccapy.spc_jog_vel.increase	HAL_BIT IN	as long as True the value of the slider will increase
gmoccapy.spc_jog_vel.decrease	HAL_BIT IN	as long as True the value of the slider will decrease
gmoccapy.spc_jog_vel.scale	HAL_FLOAT IN	A value to scale the output value (Handy to change units/min to units/sec)
gmoccapy.spc_jog_vel.value	HAL_FLOAT OUT	value of the widget

PIN	TYPE	FUNCTION
gmoccapy.spc_jog_vel.scaled-value	HAL_FLOAT OUT	Scaled value of the widget
FEED		
gmoccapy.spc_feed.increase	HAL_BIT IN	as long as True the value of the slider will increase
gmoccapy.spc_feed.decrease	HAL_BIT IN	as long as True the value of the slider will decrease
gmoccapy.spc_feed.scale	HAL_FLOAT IN	A value to scale the output value (Handy to change units/min to units/sec)
gmoccapy.spc_feed.value	HAL_FLOAT OUT	Value of the widget
gmoccapy.spc_feed.scaled-value	HAL_FLOAT OUT	Scaled value of the widget
SPINDLE		
gmoccapy.spc_spindle.increase	HAL_BIT IN	as long as True the value of the slider will increase
gmoccapy.spc_spindle.decrease	HAL_BIT IN	as long as True the value of the slider will decrease
gmoccapy.spc_spindle.scale	HAL_FLOAT IN	A value to scale the output value (Handy to change units/min to units/sec)
gmoccapy.spc_spindle.value	HAL_FLOAT OUT	Value of the widget
gmoccapy.spc_spindle.scaled-value	HAL_FLOAT OUT	Scaled value of the widget
RAPIDS		
gmoccapy.spc_rapid.increase	HAL_BIT IN	as long as True the value of the slider will increase
gmoccapy.spc_rapid.decrease	HAL_BIT IN	as long as True the value of the slider will decrease
gmoccapy.spc_rapid.scale	HAL_FLOAT IN	A value to scale the output value (Handy to change units/min to units/sec)
gmoccapy.spc_rapid.value	HAL_FLOAT OUT	Value of the widget
gmoccapy.spc_rapid.scaled-value	HAL_FLOAT OUT	Scaled value of the widget

The float pin do accept values from 0.0 to 1.0, being the percentage value you want to set the slider value.



Warning

If you use both connection types, do not connect the same slider to both pin, as the influences between the two has not been tested! Different sliders may be connected to the one or other HAL connection type.



Important

Please be aware, jog velocity depends on the turtle button state, it will lead to different slider scales depending on the mode (turtle or rabbit). Please take also a look to [jog velocities and turtle-jog HAL pin](#) for more details.

Example

Spindle Override Min Value = 20 %

```
Spindle Override Max Value = 120 %
gmoccapypy.analog-enable = 1
gmoccapypy.spindle-override-value = 0.25
```

```
value to set = Min Value + (Max Value - Min Value) * gmoccapypy.spindle-override-value
value to set = 20 + (120 - 20) * 0.25
value to set = 45 %
```

5.2.5.3 Jog HAL Pins

All axes given in the INI file have a jog-plus and a jog-minus pin, so hardware momentary switches can be used to jog the axes.

Note

naming of this HAL pin has changed for GMOCCAPY2

For the standard XYZ config following HAL Pin will be available:

- gmoccapypy.jog.axis.jog-x-plus
- gmoccapypy.jog.axis.jog-x-minus
- gmoccapypy.jog.axis.jog-y-plus
- gmoccapypy.jog.axis.jog-y-minus
- gmoccapypy.jog.axis.jog-z-plus
- gmoccapypy.jog.axis.jog-z-minus

If you use a 4 axes INI file, there will be two additional pins

- gmoccapypy.jog.jog-<your fourth axis letter >-plus
- gmoccapypy.jog.jog-<your fourth axis letter >-minus

For a "C" axis you will see:

- gmoccapypy.jog.axis.jog-c-plus
- gmoccapypy.jog.axis.jog-c-minus

5.2.5.4 Jog Velocities And Turtle-Jog HAL Pin

The jog velocity can be selected with the corresponding slider. The scale of the slider will be modified if the turtle button (the one showing a rabbit or a turtle) has been toggled. If the button is not visible, it might have been disabled on the [settings page](#). If the button shows the rabbit-icon, the scale is from min to max machine velocity. If it shows the turtle, the scale will reach only 1/20 of max velocity by default. The used divider can be set on the [settings page](#).

So using a touch screen it is much easier to select smaller velocities.

GMOCCAPY offers a HAL pin to toggle between turtle and rabbit jogging

- gmoccapypy.jog.turtle-jog (HAL Bit In)
-

5.2.5.5 Jog Increment HAL Pins

The jog increments are selectable through HAL pins, so a selection hardware switch can be used to select the increment to use. There will be a maximum of 10 HAL pin for the increments given in the INI file, if you give more increments in your INI file, they will be not reachable from the GUI as they will not be displayed.

If you have 6 increments in your HAL you will get 7 pins: jog-inc-0 is unchangeable and will represent continuous jogging.

- gmoccapj.jog.jog-inc-0
- gmoccapj.jog.jog-inc-1
- gmoccapj.jog.jog-inc-2
- gmoccapj.jog.jog-inc-3
- gmoccapj.jog.jog-inc-4
- gmoccapj.jog.jog-inc-5
- gmoccapj.jog.jog-inc-6

GMOCAPY offers also a HAL pint to output the selected jog increment

- gmoccapj.jog.jog-increment

5.2.5.6 Hardware Unlock Pin

To be able to use a key switch to unlock the settings page the following pin is exported.

- gmoccapj.unlock-settings

The settings page is unlocked if the pin is high. To use this pin, you need to activate it on the settings page.

5.2.5.7 Error/Warning Pins

- gmoccapj.error *BIT OUT*
- gmoccapj.delete-message *BIT IN*
- gmoccapj.warning-confirm *BIT IN* Confirms warning dialog like click on OK

gmoccapj.error is an bit out pin, to indicate an error, so a light can lit or even the machine may be stopped. It will be reset with the pin gmoccapj.delete-message. gmoccapj.delete-message will delete the first error and reset the gmoccapj.error pin to False after the last error has been cleared.

Note

Messages or user infos will not affect the gmoccapj.error pin, but the gmoccapj.delete-message pin will delete the last message if no error is shown!

5.2.5.8 User Created Message HAL Pins

GMOCCAPY may react to external errors, using 3 different user messages: All are HAL_BIT pin.

Status

- gmoccap.messages.statustest

Yesnodialog

- gmoccap.messages.yesnodialog
- gmoccap.messages.yesnodialog-waiting
- gmoccap.messages.yesnodialog-response

Okdialog

- gmoccap.messages.okdialog
- gmoccap.messages.okdialog-waiting

To add user created message you need to add the message to the INI file in the [DISPLAY] section. Here are a couple of examples.

```
MESSAGE_BOLDTEXT = LUBE SYSTEM FAULT
MESSAGE_TEXT = LUBE FAULT
MESSAGE_TYPE = okdialog
MESSAGE_PINNAME = lube-fault
```

```
MESSAGE_BOLDTEXT = NONE
MESSAGE_TEXT = X SHEAR PIN BROKEN
MESSAGE_TYPE = status
MESSAGE_PINNAME = pin
```

To *connect* new pins to and input you need to do this in the postgui HAL file. Here are some example connections that have the signal connected to an input some place else in the HAL file.

```
net gmoccap-lube-fault gmoccap.messages.lube-fault
net gmoccap-lube-fault-waiting gmoccap.messages.lube-fault-waiting
net gmoccap-pin gmoccap.messages.pin
```

For more information on HAL files and the net command see the [HAL Basics](#).

5.2.5.9 Spindle Feedback Pins

There are two pins for spindle feedback

- gmoccap.spindle_feedback_bar
- gmoccap.spindle_at_speed_led

gmoccap.spindle_feedback_bar will accept an float input to show the spindle speed. *gmoccap.spindle_at_speed_led* is an bit-in-pin to lit the GUI led if spindle is at speed.

5.2.5.10 Pins To Indicate Program Progress Information

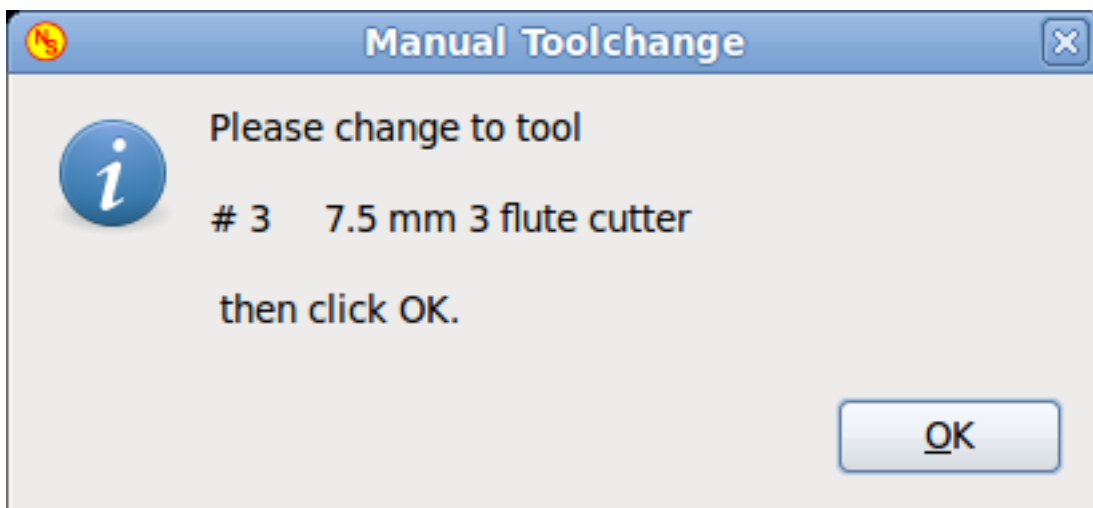
There are three pins giving information over the program progress

- gmoccapy.program.length HAL_S32 showing the total number of lines of the program
- gmoccapy.program.current-line HAL_S32 indicating the current working line of the program
- gmoccapy.program.progress HAL_FLOAT giving the program progress in percentage

The values may not be very accurate, if you are working with subroutines or large remap procedures, also loops will cause different values.

5.2.5.11 Tool related pins

Tool Change Pins These pins are provided to use GMOCCAPY's internal tool change dialog, similar to the one known from axis, but with several modifications. So you will not only get the message to change to *tool number 3*, but also the description of that tool like *7.5 mm 3 flute cutter*. The information is taken from the tool table, so it is up to you what to display.



- gmoccapy.toolchange-number S32 IN The number of the tool to be changed
- gmoccapy.toolchange-change BIT IN Indicates that a tool has to be changed
- gmoccapy.toolchange-changed BIT OUT Indicates tool has been changed
- gmoccapy.toolchange-confirm BIT IN Confirms tool change

Usually they are connected like this for a manual tool change:

```
net tool-change gmoccapy.toolchange-change <= iocontrol.0.tool-change
net tool-changed gmoccapy.toolchange-changed <= iocontrol.0.tool-changed
net tool-prep-number gmoccapy.toolchange-number <= iocontrol.0.tool-prep-number
net tool-prep-loop iocontrol.0.tool-prepare <= iocontrol.0.tool-prepared
```

Note

Please take care, that this connections have to be done in the postgui HAL file!

Tool Offset Pins These pins allow you to show the active tool offset values for X and Z in the tool information frame. You should know that they are only active after G43 has been sent.

Tool information			
Tool no.	Diameter	offset z	offset x
1	0,4000	0.017	1.161
	60 Grad vorn		

- gmoccapy.tooloffset-x
- gmoccapy.tooloffset-z

Note

Please take care, that this connections have to be done in the postgui HAL file!

Note

The tooloffset-x line is not needed on a mill, and will not be displayed on a mill with trivial kinematics.

```
net tooloffset-x gmoccapy.tooloffset-x <= motion.tooloffset.x
net tooloffset-z gmoccapy.tooloffset-z <= motion.tooloffset.z
```

Please note, that GMOCCAPY takes care of its own to update the offsets, sending an G43 after any tool change, **but not in auto mode!**



Important

So writing a program makes you responsible to include an G43 after each tool change!

5.2.6 Auto Tool Measurement

GMOCCAPY offers an integrated auto tool measurement. To use this feature, you will need to do some additional settings and you may want to use the offered HAL pin to get values in your own ngc remap procedure.



Important

Before starting the first test, do not forget to enter the probe height and probe velocities on the settings page! See [Settings Page Tool Measurement](#)

It might be also a good idea to take a look at the tool measurement video: see [tool measurement related videos](#)

Tool Measurement in GMOCCAPY is done a little bit different to many other GUI. You should follow these steps:

- touch of you workpiece in X and Y
-

- measure the height of your block from the base where your tool switch is located, to the upper face of the block (including chuck etc.)
- Push the button block height and enter the measured value
- Go to auto mode and start your program

here is a small sketch:

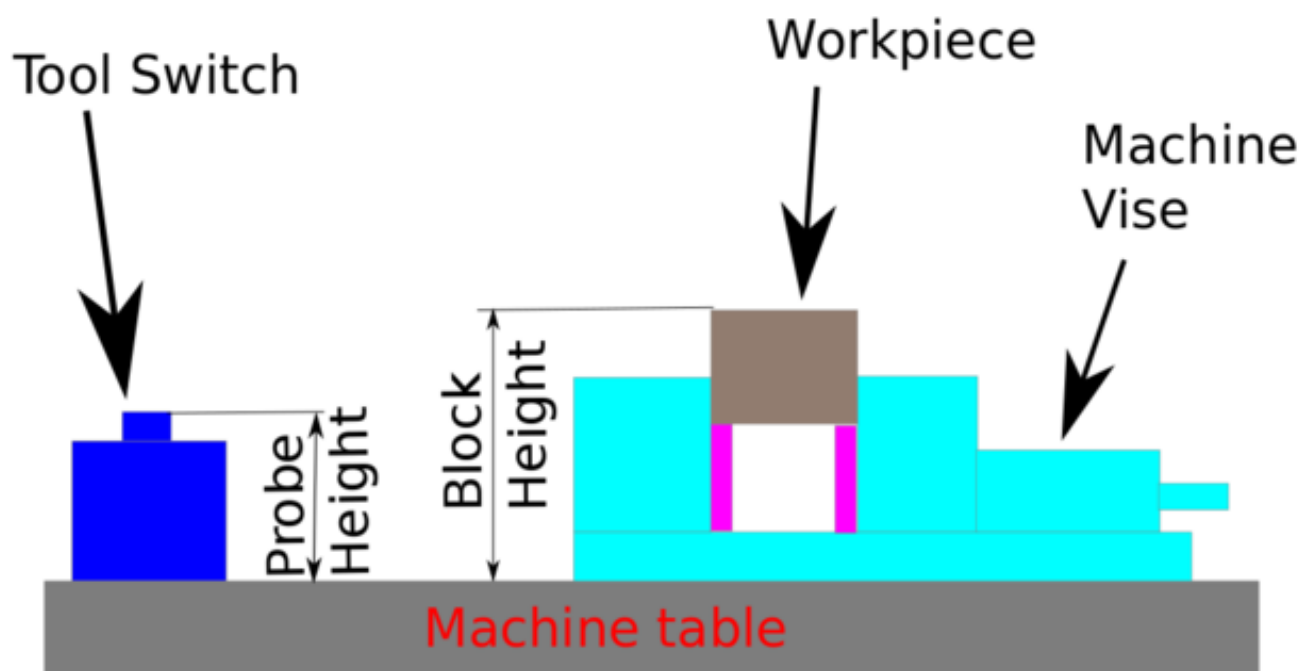


Figure 5.19: Tool measurement data

With the first given tool change the tool will be measured and the offset will be set automatically to fit the block height. The advantage of the GMOCCAPY way is, that you do not need a reference tool.

Note

Your program must contain a tool change at the beginning! The tool will be measured, even it has been used before, so there is no danger, if the block height has changed. There are several videos showing the way to do that on you tube.

5.2.6.1 Tool Measurement Pins

GMOCCAPY offers 5 pins for tool measurement purpose. The pins are mostly used to be read from a G-code subroutine, so the code can react to different values.

- gmoccap.toolmeasurement HAL_BIT enable or not tool measurement
- gmoccap.blockheight HAL_FLOAT the measured value of the top face of the workpiece
- gmoccap.probeheight HAL_FLOAT the probe switch height
- gmoccap.searchvel HAL_FLOAT the velocity to search for the tool probe switch
- gmoccap.probevel HAL_FLOAT the velocity to probe tool length

5.2.6.2 Tool Measurement INI File Modifications

Modify your INI file to include the following:

The RS274NGC section

```
[RS274NGC]
# is the sub, with is called when a error during tool change happens, not needed on every machine configuration ↵
ON_ABORT_COMMAND=0 <on_abort> call

# The remap code
REMAP=M6 modalgroup=6 prolog=change_prolog ngc=change epilg=change_epilog
```

Note

Make sure INI_VARS and HAL_PIN_VARS are not set to 0. They are set to 1 by default.

The Tool Sensor Section The position of the tool sensor and the start position of the probing movement, all values are absolute coordinates, except MAXPROBE, what must be given in relative movement.

```
[TOOLSENSOR]
X = 10
Y = 10
Z = -20
MAXPROBE = -20
```

The Change Position Section This is not named TOOL_CHANGE_POSITION on purpose - **canon uses that name and will interfere otherwise**. The position to move the machine before giving the change tool command. All values are in absolute coordinates.

```
[CHANGE_POSITION]
X = 10
Y = 10
Z = -2
```

The Python Section The Python plug ins serves interpreter and task.

```
[PYTHON]
# The path to start a search for user modules
PATH_PREPEND = python
# The start point for all.
TOPLEVEL = python/toplevel.py
```

5.2.6.3 Needed Files

You must copy the following files to your config directory

First make a directory *python* in your config folder from *your_linuxcnc-dev_directory/configs/sim/gmoccapy/* copy *toplevel.py* to your *config_dir/python* folder. Copy *remap.py* to your *config_dir/python* folder. Copy *stdglue.py* to your *config_dir/python* folder.

From *your_linuxcnc-dev_directory/configs/sim/gmoccapy/macros* copy *on_abort.ngc* to the directory specified in the SUBROUTINE_PATH, see [RS274NGC Section](#). From *your_linuxcnc-dev_directory/configs/sim/gmoccapy/macros* copy *change.ngc* to the directory specified as SUBROUTINE_PATH, see [RS274NGC Section](#).

Open *change.ngc* with a editor and uncomment the following lines (49 and 50):

```
F #<_hal[gmoccapy.probevel]>
G38.2 Z-4
```

You may want to modify this file to fit more your needs.

5.2.6.4 Needed HAL Connections

Connect the tool probe in your HAL file like so:

```
net probe motion.probe-input <= <your_input_pin>
```

The line might look like this:

```
net probe motion.probe-input <= parport.0.pin-15-in
```


In your postgui.hal file add:

```
# The next lines are only needed if the pins had been connected before
unlinkp iocontrol.0.tool-change
unlinkp iocontrol.0.tool-changed
unlinkp iocontrol.0.tool-prep-number
unlinkp iocontrol.0.tool-prepared

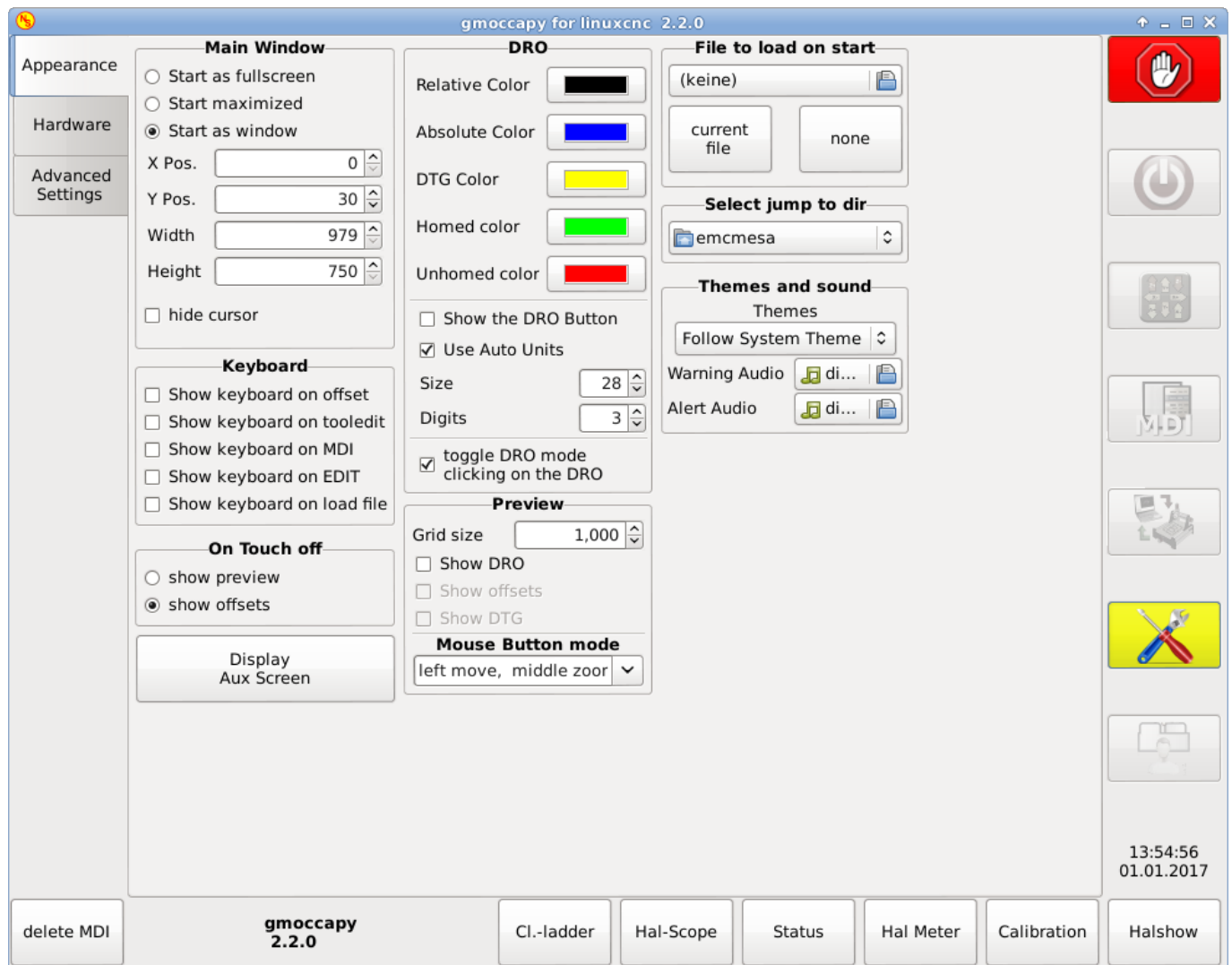
# link to GMOCCAPY toolchange, so you get the advantage of tool description on change ↔
dialog
net tool-change gmoccapy.toolchange-change <= iocontrol.0.tool-change
net tool-changed gmoccapy.toolchange-changed <= iocontrol.0.tool-changed
net tool-prep-number gmoccapy.toolchange-number <= iocontrol.0.tool-prep-number
net tool-prep-loop iocontrol.0.tool-prepare <= iocontrol.0.tool-prepared
```

5.2.7 The Settings Page



To enter the page you will have to click on  and give an unlock code, which is **123** as default. If you want to change it at this time you will have to edit the hidden preference file, see [the display section](#) for details.

The page looks at the moment like so:



The page is separated in three main tabs:

5.2.7.1 Appearance

On this tab you will find the following options:

Here you can select how you wish the GUI to start. The main reason for this was the wish to get an easy way for the user to set the starting options without the need to touch code. You have three options:

- start as full screen
- start maximized
- start as window:: If you select start as window the spinboxes to set the position and size will get active. One time set, the GUI will start every time on the place and with the size selected. Nevertheless the user can change the size and position using the mouse, but that will not have any influence on the settings.
- *hide the cursor** does allow to hide the cursor, what is very useful if you use a touch screen.

The check-boxes allows the user to select if he want the on board keyboard to be shown immediately, when entering the MDI Mode, when entering the offset page, the tooledit widget or when open a

program in the EDIT mode. The keyboard button on the bottom button list will not be affected by this settings, so you be able to show or hide the keyboard by pressing the button. The default behavior will be set by the check-boxes.

Default are :

Note

If this section is not sensitive, you have not installed a virtual keyboard, supported are *onboard* and *matchbox-keyboard*.

- show keyboard on offset = True
- show keyboard on tooledit = False
- show keyboard on MDI = True
- show keyboard on EDIT = True
- show keyboard on load file = False

If the keyboard layout is not correct, i.e. clicking X gives Z, than the layout has not been set properly, related to your locale settings. For onboard it can be solved with a small batch file with the following content:

```
#!/bin/bash
setxkbmap -model pc105 -layout de -variant basic
```

The letters "de" are for German, you will have to set them according to your locale settings. Just execute this file before starting LinuxCNC, it can be done also adding a starter to your local folder.

```
./config/autostart
```

So that the layout is set automatically on starting.

For matchbox-keyboard you will have to make your own layout, for a German layout ask in the forum. give the option to show the preview tab or the offset page tab if you enter the touch off mode by clicking the corresponding bottom button.

- show preview
- show offsets

As the notebook tabs are shown, you are able to switch between both views in any case.

Clicking this button will open an additional window. This button is only sensitive if a file named *Gmoccapy 3.glade* is in your configuration folder. is in your configuration folder. You can build the Aux screen using Glade.

**Warning**

The main window of the auxiliary screen must be named *window2*.

You have the option to select the background colors of the different DRO states. So users suffering from protanopia (red/green weakness) are able to select proper colors.

By default, the backgrounds are:

- Relative mode = black
- Absolute mode = blue
- Distance to go = yellow

The foreground color of the DRO can be selected with:

- homed color = green
- unhomed color = red

Options on data to display:

show dro

The DRO will be shown in the preview window.

show offsets

The offsets will be shown in the preview window.

show DTG

The distance to go will be shown in the preview window.

Note

You can change through the DRO modes (absolute, relative, distance to go) by clicking on the DRO! **if you click on the left side letter of the DRO a popup window will allow you to set the value of the axis, making it easier to set the value, as you will not need to go over the touch off bottom button. Clicking the numbers (right side of the DRO) will toggle through the DRO modes as described above.**

size

Allows to set the size of the DRO font, default is 28, if you use a bigger screen you may want to increase the size up to 56. If you do use 4 axis, the DRO font size will be 3/4 of the value, because of space reason.

digits

Sets the number of digits of the DRO from 1 to 5.

Note

Imperial will show one digit more than metric. So if you are in imperial machine units and set the digit value to 1, you will get no digit at all in metric.

toggle DRO mode

If not active, a mouse click on the DRO will not take any action.

By default this checkbox is active, so every click on any DRO will toggle the DRO readout from actual to relative to DTG (distance to go).

Nevertheless a click on the axis letter will open the popup dialog to set the axis value.

Grid Size Sets the grid size of the preview window. Unfortunately the size **has to be set in inches**, even if your machine units are metric. We do hope to fix that in a future release.

Note

The grid will not be shown in perspective view.

Show DRO Will show the a DRO also in the preview window, it will be shown automatically in fullsize preview

Show DTG will show also the DTG (direct distance to end point) in the preview, only if Show DRO is active and not full size preview.

Show Offsets will show the offsets in the preview window.

Note

If you only check this option and leave the others unchecked, you will get in full size preview a offset page

Mouse Button Mode this combobox you can select the button behavior of the mouse to rotate, move or zoom within the preview:

- left rotate, middle move, right zoom
- left zoom, middle move, right rotate
- left move, middle rotate, right zoom
- left zoom, middle rotate, right move
- left move, middle zoom, right rotate
- left rotate, middle zoom, right move

Default is left move, middle zoom, right rotate.

The mouse wheel will still zoom the preview in every mode.

Tip

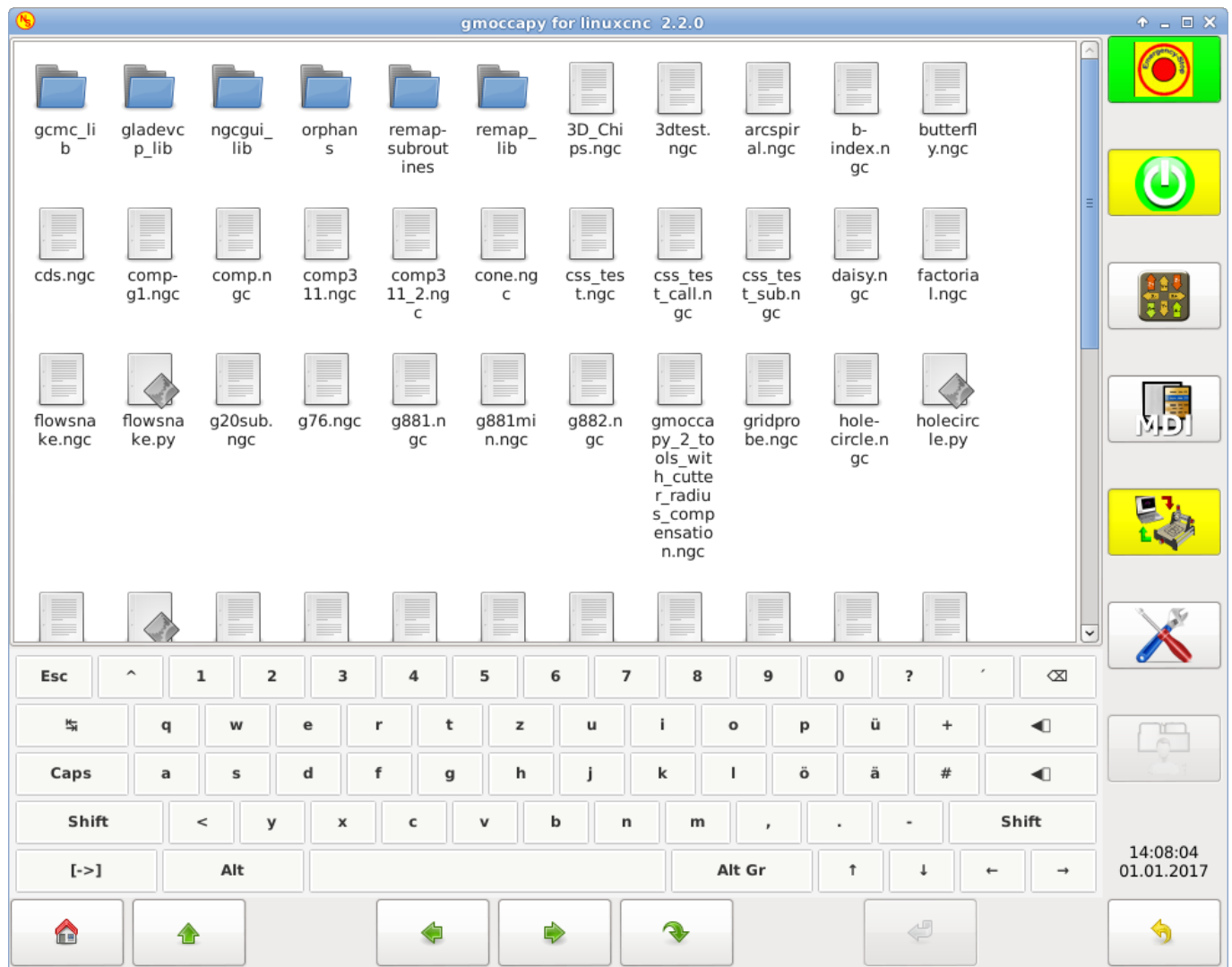
If you select an element in the preview, the selected element will be taken as rotation center point and in auto mode the corresponding code line will be highlighted.

Select the file you want to be loaded on start up. In other GUI changing this was very cumbersome, because the users where forced to edit the INI file.

Select the file you want to be loaded on start up. If a file is loaded, it can be set by pressing the current button to avoid that any program is loaded at start up, just press the None button.

The file selection screen will use the filters you have set in the INI file, if there aren't any filters given, you will only see **ngc** files. The path will be set according to the INI settings in [DISPLAY] PROGRAM_PREFIX.

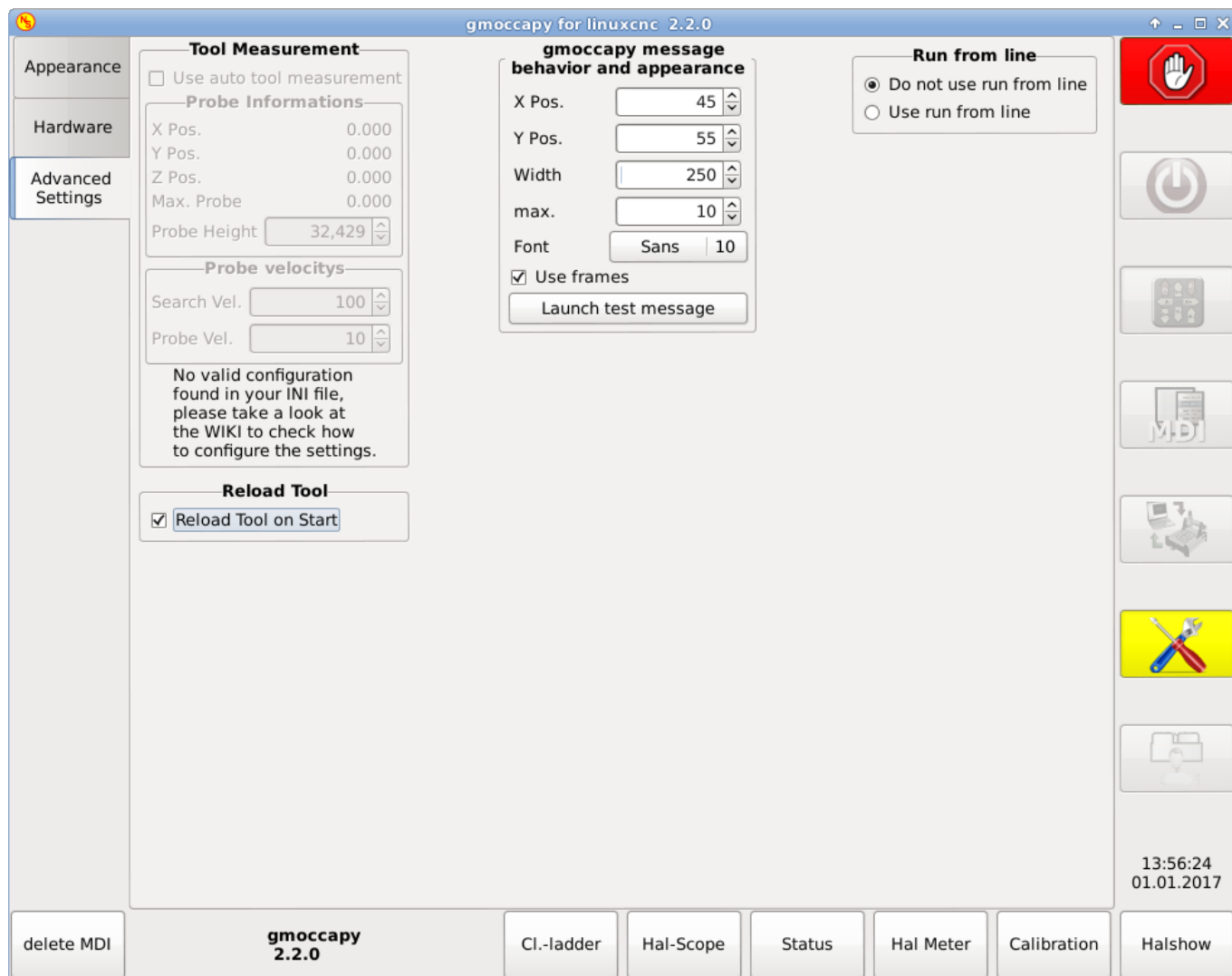
You can set here the directory to jump to if you press the corresponding button in the file selection dialog.



This lets the user select what desktop theme to apply and what error and messages sounds should be played. By default "Follow System Theme" is set.

See section [Icon Theme](#) for details.

5.2.7.2 Hardware



For the different HAL Pin to connect MPG Wheels to, you may select individual scales to be applied. The main reason for this was my own test to solve this through HAL connections, resulting in a very complex HAL file. Imagine a user having an MPG Wheel with 100 ipr and he wants to slow down the max. vel. from 14000 to 2000 mm/min, that needs 12000 impulses, resulting in 120 turns of the wheel! Or an other user having a MPG Wheel with 500 ipr and he wants to set the spindle override witch has limits from 50 to 120 % so he goes from min to max within 70 impulses, meaning not even 1/4 turn.

By default all scales are set using the calculation:

$$(MAX - MIN)/100$$

Some users want to jog there machine using the keyboard buttons and there are others that will never allow this. So everybody can select whether to use them or not. It is not recommended to use keyboard jogging, as it represents a serious risk for operator and machine.

Default is not to use keyboard shortcuts.

Please take care if you use a lathe, than the shortcuts will be different, see [the Lathe section](#).

- Arrow Left or NumPad_Left = X minus

- Arrow Right or NumPad_Right = X plus
- Arrow up or NumPad_Up = Y plus
- Arrow Down or NumPad_Down = Y minus
- Page Up or NumPad_Page_Up = Z plus
- Page Down or NumPad_Page_Down = Z minus
- F1 = Estop (will work even if keyboard shortcuts are disabled)
- F2 = Machine on
- F3 = manual mode
- F5 = MDI mode
- ESC = Abort

In AUTO Mode we will allow the following key shortcuts * R or r = run program * P or p = pause program * S or s = resume program * Control and R or r will reload the loaded file

There are additional keys for message handling, see [Message behavior and appearance](#)

- WINDOWS = Delete last message
- <CTRL><SPACE> = Delete all messages

There are three options to unlock the settings page:

- use unlock code (the user must give a code to get in)
- Do not use unlock code (There will be no security check)
- Use HAL pin to unlock (hardware pin must be high to unlock the settings, see [hardware unlock pin](#))

Default is use unlock code (default = **123**)

The start RPM sets the rpm to be used if the spindle is started and no S value has been set.

Note

This value will be presetted according to your settings in [DISPLAY] DEFAULT_SPINDLE_SPEED of your INI. If you change the settings on the settings page, that value will be default from that moment, your INI file will not be modified.

With the MIN and MAX settings you set the limits of the spindle bar shown in the INFO frame on the main screen. It is no error giving wrong values. If you give a maximum of 2000 and your spindle makes 4000 rpm, only the bar level will be wrong on higher speeds than 2000 rpm.

```
default values are
MIN = 0
MAX = 6000
```

Turtle Jog

This settings will have influence on the jog velocities.

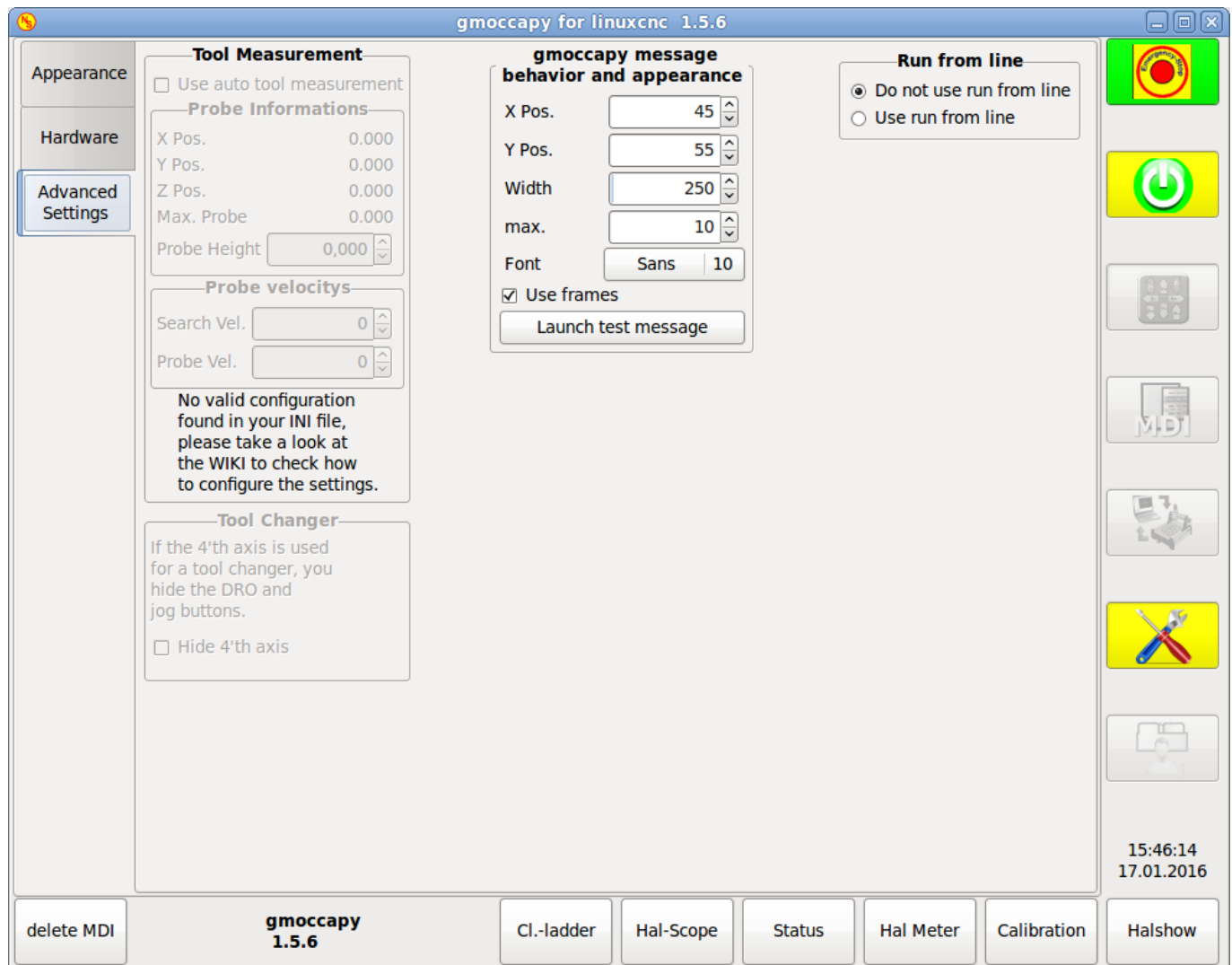
- *hide turtle jog button* will hide the button right of the jog velocity slider, if you hide this button, please take care that it shows the rabbit icon, otherwise you will not be able to jog faster than the turtle jog velocity, which is calculated using the turtle jog factor.
-

- *Turtle jog factor* sets the scale to apply for turtle jog mode. If you set a factor of 20, the turtle max jog velocity will be 1/20 of max. velocity of the machine if in turtle mode (button pressed, showing the turtle).

Note

This button can be activated using the [turtle-jog](#) HAL pin.

5.2.7.3 Advanced Settings



Note

If this part is not sensitive, you do not have a valid INI file configuration to use tool measurement.

Please check [Auto Tool Measurement](#)

- Use auto tool measurement : If checked, after each tool change, a tool measurement will be done, the result will be stored in the tool table and an G43 will be executed after the change..

The following information is taken from your INI file and must be given in absolute coordinates.

- X Pos. = The X position of the tool switch
- Y Pos. = The Y position of the tool switch
- Z Pos. = The Z position of the tool switch, we will go as rapid move to this coordinate
- Max. Probe = is the distance to search for contact, an error will be launched, if no contact is given. The distance has to be given in relative coordinates, beginning the move from Z Pos., so you have to give a negative value to go down!
- Probe Height = is the height of your probe switch, you can measure it. Just touch off the base where the probe switch is located and set that to zero. Then make a tool change and watch the tool_offset_z value, that is the height you must enter here.
- Search Vel. = The velocity to search for the tool switch, after contact the tool will go up again and then goes toward the probe again with probe vel, so you will get better results.
- Probe Vel. = Is the velocity for the second movement to the switch, it should be slower to get better touch results. (In sim mode, this is commented out in macros/change.ngc, otherwise the user would have to click twice on the probe button.)

If your fourth axis is used as a tool changer, you may want to hide the DRO and all other buttons related to that axis.

You can do that by marking the checkbox, which will hide:

- 4th axis DRO
- 4th axis jog button
- 4th axis homing button
- 4th axis column on the offset page.
- 4th axis column in the tool editor.

If checked, the tool in spindle will be saved on each change in the preference file, making it possible to reload the last mounted tool on start up. The tool will be loaded after all axis are homed, because before it is not allowed to execute MDI commands. If you use NO_FORCE_HOMING you can not use this feature, because the needed all_homed_signal will never be emitted.

This will display small pop up windows displaying the message or error text, the behavior is very similar to the one axis uses. You can delete a specific message, by clicking on it's close button, if you want to delete the last one, just hit the WINDOWS key on your keyboard, or delete all messages at ones with <CTRL><SPACE>.

You are able to set some options:

- X Pos = The position of the top left corner of the message in X counted in pixel from the top left corner of the screen.
 - Y Pos = The position of the top left corner of the message in Y counted in pixel from the top left corner of the screen.
 - Width = The width of the message box
 - max = The maximum messages you want to see at ones, if you set this to 10, the 11th message will delete the first one, so you will only see the last 10 ones.
 - Font = The font and size you want to use to display the messages
 - use frames = If you activate the checkbox, each message will be displayed in a frame, so it is much easier to distinguish the messages. But you will need a little bit more space.
-

- The button launch test message will just do what it is supposed to, it will show a message, so you can see the changes of your settings without the need to generate an error.

You can allow or disallow the run from line. This will set the corresponding button insensitive (grayed out), so the user will not be able to use this option. The default is disable run from line.



Warning

It is not recommend to use run from line, as LinuxCNC will not take care of any previous lines in the code before the starting line. So errors or crashes are very probable.

5.2.8 Icon Theme

Icon themes are used to customize the look and feel of GMOCCAPY's icons.

GMOCCAPY ships with three different icon themes:

- classic: The classic GMOCCAPY icons
- material: A modern icon theme inspired by Google's Material Icons that automatically adopts its coloring from the selected desktop theme.
- material-light: Derived from material but optimized for light desktop themes.

The icon theme used in GMOCCAPY is a regular GTK IconTheme that follows the freedesktop icon theme specification. Thus every valid GTK IconTheme can be used as GMOCCAPY icon theme as long as it contains the required icons.

GMOCCAPY scans the following directories for IconThemes:

- linuxcnc/share/gmoccapy/icons
- ~/.icons

5.2.8.1 Custom Icon Theme

Creating a custom IconTheme is pretty easy. All you need is a text editor and of course the desired icons as pixel or vector graphics. Details about how exactly an IconTheme is built can be found at [Freedesktop: Icon Theme Specification](#)

Start by creating an empty directory with the name of the icon theme. Place the directory in one of GMOCCAPY's IconTheme directories. Then we need a file called index.theme in the root folder of our icon theme which contains the required metadata for the theme. That's a simple text file with at least the following sections:

```
[Icon Theme]
Name=YOUR_THEME_NAME
Comment=A DESCRIPTION OF YOUR THEME
Inherits=hicolor
Directories=16x16/actions,24x24/actions,32x32/actions,48x48/actions,scalable/ ←
actions
```

- Name: The name of your icon theme
- Comment: A description of your icon theme

- Inherits: A icon theme can derive from another icon theme, the default is hicolor
- Directories: A comma separated list of all the directories of your icon theme

Each directory usually contains all the icons of the theme in a specific size, for example 16x16/actions should contain all icons with the category "actions" in the size 16x16 pixels as pixel-graphics (e.g. png files). A special case is the directory called "scalable/actions", this contains scalable icons not tied to a specific size (e.g. svg files).

By supplying different sized versions of the icons, we can guarantee a nice looking icon if different sizes and we also have the ability to change the icon according to its size, for example a 64x64 px sized icon may contain more details than its 16x16 px version.

For each directory we also have to write a section in the index.theme file:

```
[16x16/actions]
Size=16
Type=Fixed
Context=Actions

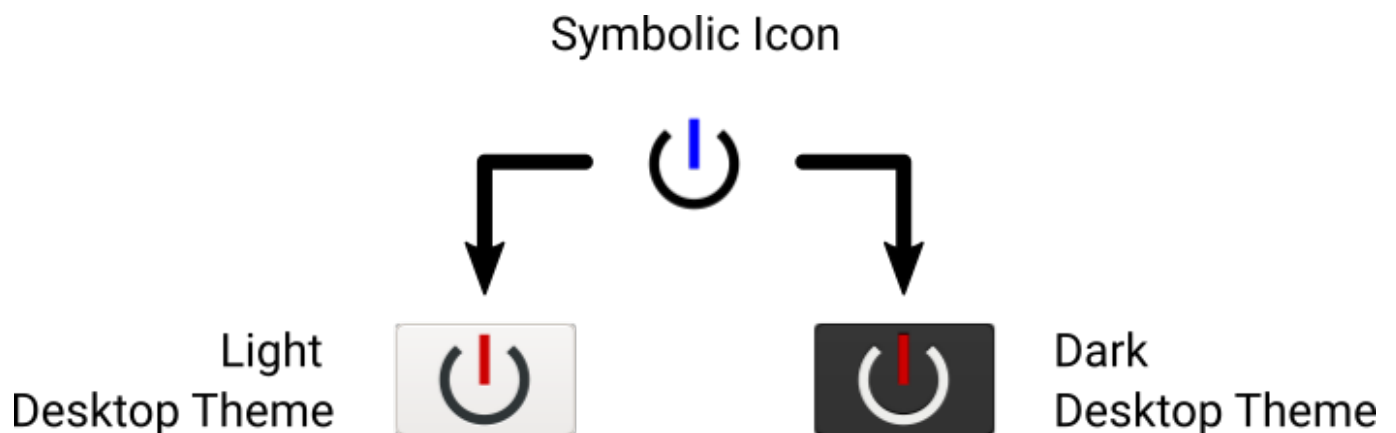
[scalable/actions]
Size=48
Type=Scalable
Context=Actions
```

- Size: Nominal icon size in this directory
- Type: Fixed, Threshold or Scalable
- Context: Intended "category" of icons

Basically that's everything needed to create a custom IconTheme.

5.2.8.2 Symbolic Icons

Symbolic icons are a special type of icon, usually a monochrome image. The special feature of symbolic icons is that the icons are automatically colored at runtime to match the desktop theme. That way, icon themes can be created that work well with dark and also light desktop themes (in fact, that's not always the best option, that's why a dedicated "material-light" theme exists).



To make use of the symbolic feature, a icon file has to have the suffix `.symbolic.$ext` (where `$ext` is the regular file extension like `png`) for example `"power_on.symbolic.png"`.

With that name, GTK treats this image as symbolic icon and applies some recoloring when loading the icon. There are only four colors allowed to use:

Color	Hex Code	Description
black	#000000	Primary color, gets changed to match the desktop themes primary color
red	#ff0000	Success: this color indicates "success" (usually somethin green'ish)
green	#00ff00	Warning: this color indicates "warning" (usually somethin yellow/orange'ish)
blue	#0000ff	Error: this color indicates "error" (usually somethin red'ish)

Tip

Examples of symbolic icons can be found at [linuxcnc/share/gmoccapy/icons/material](https://linuxcnc.org/share/gmoccapy/icons/material) Theme

5.2.9 Lathe Specific Section

If in the INI file `LATHE = 1` is given, the GUI will change its appearance to the special needs for a lathe. Mainly the Y axis will be hidden and the jog buttons will be arranged in a different order.



Figure 5.20: Normal Lathe



Figure 5.21: Back Tool Lathe

As you see the R DRO has a black background and the D DRO is gray. This will change according to the active G-code G7 or G8. The active mode is visible by the black background, meaning in the shown images G8 is active.

The next difference to the standard screen is the location of the Jog Button. X and Z have changed places and Y is gone. You will note that the X+ and X- buttons changes there places according to normal or back tool lathe.

Also the keyboard behavior will change:

Normal Lathe:

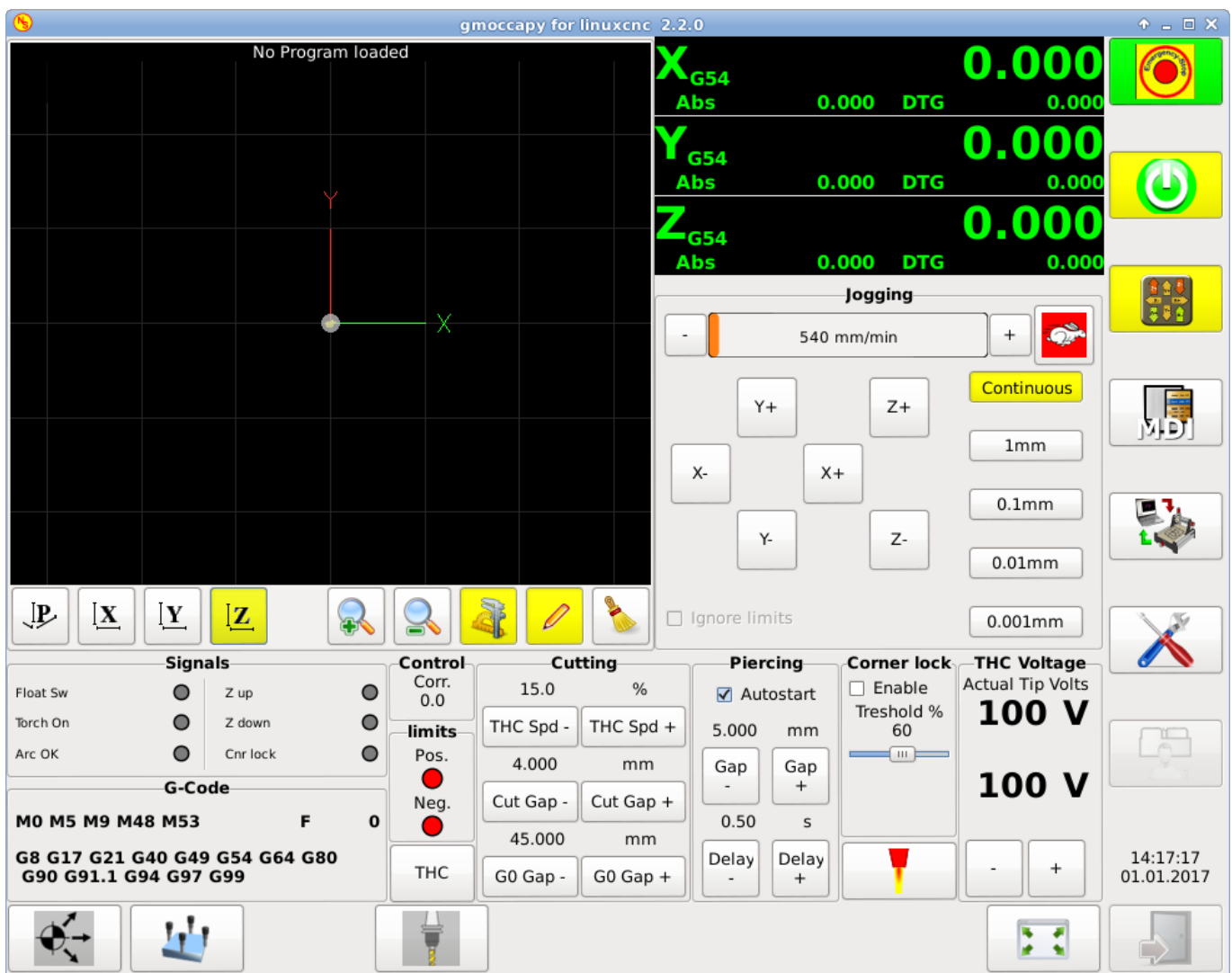
- Arrow Left or NumPad_Left = Z minus
- Arrow Right or NumPad_Right = Z plus
- Arrow up or NumPad_Up = X minus
- Arrow Down or NumPad_Down = X plus

Back Tool Lathe:

- Arrow Left or NumPad_Left = Z minus
- Arrow Right or NumPad_Right = Z plus
- Arrow up or NumPad_Up = X plus
- Arrow Down or NumPad_Down = X minus

The tool information frame will show not only the Z offset, but also the X offset and the tool table is showing all lathe relevant information.

5.2.10 Plasma Specific Section



There is a very good WIKI, which is actually growing, maintained by Marius see [Plasma wiki page](#)

5.2.11 Videos On YouTube

Below are a series of videos list that show GMOCCAPY in action. Unfortunately, these videos don't show the latest version of GMOCCAPY, but the way to use it will still be the same as in the current version. I will update the videos as soon as possible.

5.2.11.1 Basic Usage

<https://www.youtube.com/watch?v=O5B-s3uiI6g>

5.2.11.2 Simulated Jog Wheels

<http://youtu.be/ag34SGxt97o>

5.2.11.3 Settings Page

<https://www.youtube.com/watch?v=AuwhSHRJoil>

5.2.11.4 Simulated Hardware Button

German = <http://www.youtube.com/watch?v=DTqhY-MfzDE>

English = <http://www.youtube.com/watch?v=ItVWJBK9WFA>

5.2.11.5 User Tabs

<http://www.youtube.com/watch?v=rG1zmeqXyZI>

5.2.11.6 Tool Measurement Videos

Auto Tool Measurement Simulation = <http://youtu.be/rrkMw6rUFdk>

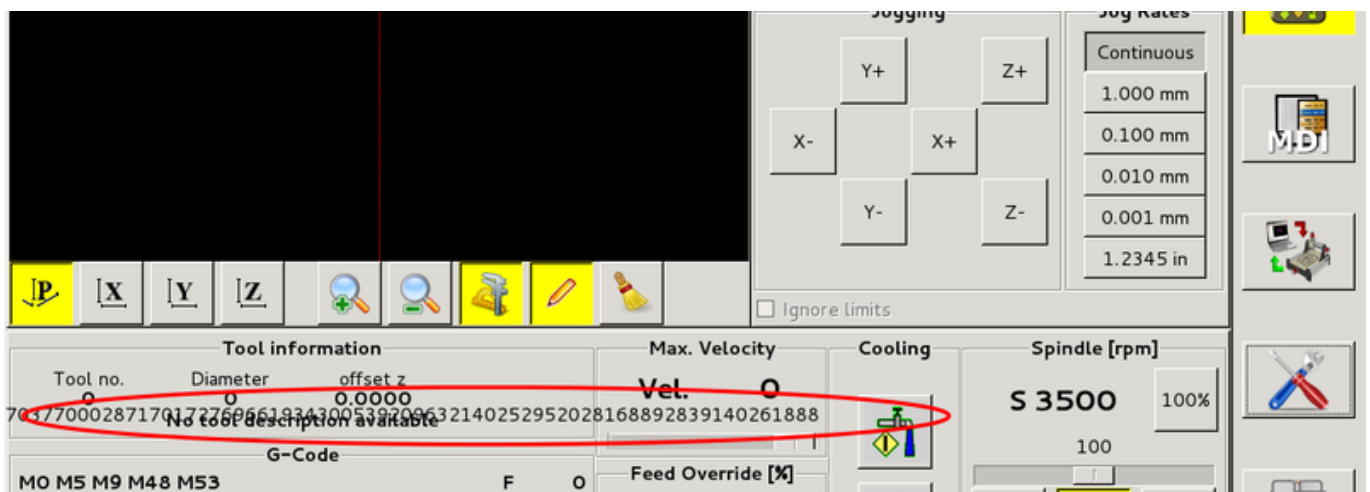
Auto Tool Measurement Screen = <http://youtu.be/Z2ULDj9dzvk>

Auto Tool Measurement Machine = <http://youtu.be/1arucCaDdX4>

5.2.12 Known problems

5.2.12.1 Strange numbers in the info area

If you get strange numbers in the info area of GMOCCAPY like:



You have made your config file using an older version of StepConfWizard. It has made a wrong entry in the INI file under the [TRAJ] named MAX_LINEAR_VELOCITY = xxx. Change that entry to MAX_VELOCITY = xxx

5.2.12.2 Not ending macro

If you use a macro without movement, like this one:

```
o<zeroxy> sub

G92.1
G92.2
G40

G10 L20 P0 X0 Y0

o<zeroxy> endsub
m2
```

GMOCCAPY will not see the end of the macro, because the interpreter needs to change its state to IDLE, but the macro does not even set the interpreter to a new state. To avoid that just add a G4 P0.1 line to get the needed signal. The correct macro would be:

```
o<zeroxy> sub

G92.1
G92.2
G40

G10 L20 P0 X0 Y0

G4 P0.1

o<zeroxy> endsub
m2
```

5.3 NGCGUI

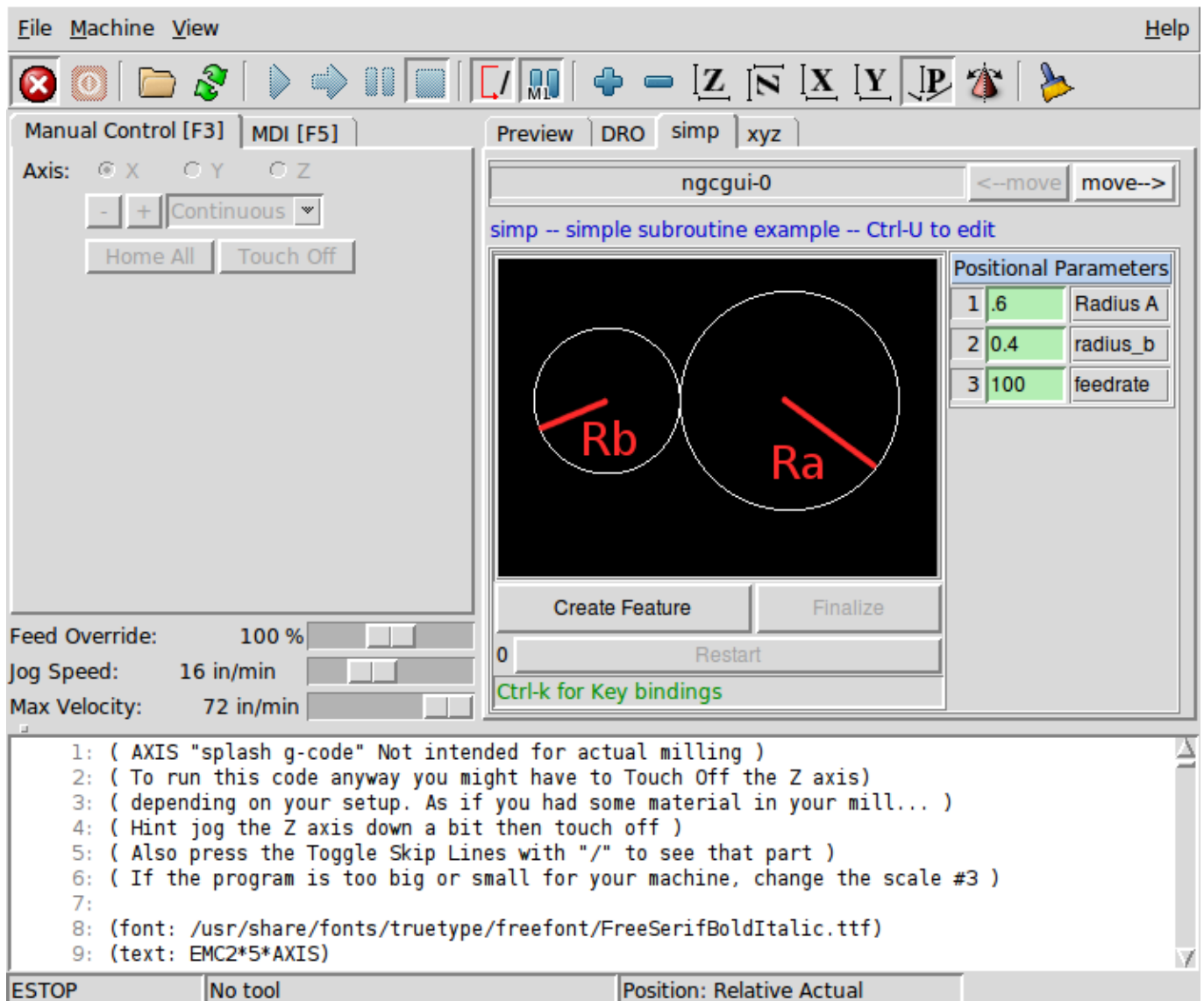


Figure 5.22: NGCGUI embedded into AXIS

5.3.1 Overview

- *NGCGUI* is a Tcl application to work with subroutines. It allows you to have a conversational interface with LinuxCNC. You can organize the subroutines in the order you need them to run and concatenate the subroutines into one file for a complete part program.
- *NGCGUI* can run as a standalone application or can be embedded in multiple tab pages in the AXIS GUI
- *PYNGCGUI* is an alternate, python implementation of ngcgui.
- *PYNGCGUI* can run as a standalone application or can be embedded as a tab page (with its own set of multiple subroutine tabs) in any GUI that supports embedding of GladeVCP applications AXIS, Touchy, gscreen and gmocappy.

Using NGCGUI or PYNGCGUI:

- Tab pages are provided for each subroutine specified in the INI file
- New subroutines tab pages can be added on the fly using the [custom tab](#)
- Each subroutine tab page provides entry boxes for all subroutine parameters
- The entry boxes can have a default value and an label that are identified by special comments in the subroutine file
- Subroutine invocations can be concatenated together to form a multiple step program
- Any single-file G-code subroutine that conforms to ngcgui conventions can be used
- Any gcmc (G-code-meta-compiler) program that conforms to ngcgui conventions for tagging variables can be used. (The gcmc executable must be installed separately, see: <http://www.vagrearg.org/content/gcmc>)

Note

NGCGUI and PYNGCGUI implement the same functions and both process .ngc and .gcmc files that conform to a few ngcgui-specific conventions. In this document, the term *NGCGUI* generally refers to either application.

5.3.2 Demonstration Configurations

A number of demonstration configurations are located in the sim directory of the Sample Configurations offered by the LinuxCNC configuration picker. The configuration picker is on the system's main menu: Applications > CNC > LinuxCNC

Examples are included for the AXIS, Touchy, gscreen, and GMOCCAPY. These examples demonstrate both 3-axis (XYZ) cartesian configurations (like mills) and lathe (XZ) setups. Some examples show the use of a pop up keyboard for touch screen systems and other examples demonstrate the use of files created for the gcmc (G-code Meta Compiler) application. The touchy examples also demonstrate incorporation of a GladeVCP back plot viewer (gremlin_view).

The simplest application is found as:

Sample Configurations/sim/axis/ngcgui/ngcgui_simple

A comprehensive example showing gcmc compatibility is at:

Sample Configurations/sim/axis/ngcgui/ngcgui_gcmc

A comprehensive example embedded as a GladeVCP app and using gcmc is at:




Sample Configurations/sim/gscreen/ngcgui/pyngcgui_gcmc

The example sim configurations make use of library files that provide example G-code subroutine (.ngc) files and G-code-meta-compiler (.gcmc) files:

- *nc_files/ngcgui_lib*
 - *ngcgui.ngc* - An easy to understand example using subroutines
 - *arc1.ngc* - basic arc using cutter radius compensation
 - *arc2.ngc* - arc speed by center, offset, width, angle (calls arc1)
 - *backlash.ngc* - routine to measure an axis backlash with dial indicator
-

- *db25.ngc* - creates a DB25 plug cutout
- *gosper.ngc* - a recursion demo (flowsnake)
- *helix.ngc* - helix or D-hole cutting
- *helix_rtheta.ngc* - helix or D-hole positioned by radius and angle
- *hole_circle.ngc* - equally spaced holes on a circle
- *ihex.ngc* - internal hexagon
- *iquad.ngc* - internal quadrilateral
- *ohex.ngc* - outside hexagon
- *oquad.ngc* - outside quadrilateral
- *qpex_mm.ngc* - demo of qpockets (mm based)
- *qpex.ngc* - demo of qpockets (inch based)
- *qpocket.ngc* - quadrilateral pocket
- *rectangle_probe.ngc* - probe a rectangular area
- *simp.ngc* - a simple subroutine example that creates two circles
- *slot.ngc* - slot from connecting two endpoints
- *xyz.ngc* - machine exerciser constrained to a box shape
- *Custom* - Creates custom tabs
- *ttt* - True Type Tracer, to create texts to be engraved
- *nc_files/ngcgui_lib/lathe*
 - *ngcgui-lathe* - Example lathe subroutine
 - *g76base.ngc* - GUI for g76 threading
 - *g76diam.ngc* - threading speced by major, minor diameters
 - *id.ngc* - bores the inside diameter
 - *od.ngc* - turns the outside diameter
 - *taper-od.ngc* - turns a taper on the outside diameter
 - *Custom* - Creates custom tabs
- *nc_files/gcmc_lib*
 - *drill.gcmc* - drill holes in rectangle pattern
 - *square.gcmc* - simple demo of variable tags for gcmc files
 - *star.gcmc* - gcmc demo illustrating functions and arrays
 - *wheels.gcmc* - gcmc demo of complex patterns

To try a demonstration, select a sim configuration and start the LinuxCNC program.

If using the AXIS GUI, press the *E-Stop*  then *Machine Power*  then *Home All*. Pick a ngcgui tab, fill in any empty blanks with sensible values and press *Create Feature* then *Finalize*. Finally press the *Run*  button to watch it run. Experiment by creating multiple features and features from different tab pages.

To create several subroutines concatenated into a single file, go to each tab fill in the blanks, press *Create Feature* then using the arrow keys move any tabs needed to put them in order. Now press *Finalize* and answer the prompt to create

Other GUIs will have similar functionality but the buttons and names may be different.

Note

The demonstration configs create tab pages for just a few of the provided examples. Any GUI with a [custom tab](#) can open any of the library example subroutines or any user file if it is in the LinuxCNC subroutine path.

To see special key bindings, click inside an ngcgui tab page to get focus and then press Control-k. The demonstration subroutines should run on the simulated machine configurations included in the distribution. A user should always understand the behavior and purpose of a program before running on a real machine.

5.3.3 Library Locations

In LinuxCNC installations installed from deb packages, the simulation configs for ngcgui use symbolic links to non-user-writable LinuxCNC libraries for:

- `nc_files/ngcgui_lib` ngcgui-compatible subfiles
- `nc_files/ngcgui_lib/lathe` ngcgui-compatible lathe subfiles
- `nc_files/gcmc_lib` ngcgui-gcmc-compatible programs
- `nc_files/ngcgui_lib/utilitysubs` Helper subroutines
- `nc_files/ngcgui_lib/mfiles` User M files

These libraries are located by INI file items that specify the search paths used by LinuxCNC (and ngcgui):

```
[RS274NGC]
SUBROUTINE_PATH = ../../nc_files/ngcgui_lib:../../nc_files/gcmc_lib:../../nc_files/ ↵
                  ngcgui_lib/utilitysubs
USER_M_PATH      = ../../nc_files/ngcgui_lib/mfiles
```

Note

These are long lines (not continued on multiple lines) that specify the directories used in a search patch. The directory names are separated by colons (:). No spaces should occur between directory names.

A user can create new directories for their own subroutines and M-files and add them to the search path(s).

For example, a user could create directories from the terminal with the commands:

```
mkdir /home/myusername/mysubs
mkdir /home/myusername/mymfiles
```

And then create or copy system-provided files to these user-writable directories. For instance, a user might create a ngcgui-compatible subfile named:

```
/home/myusername/mysubs/example.ngc
```

To use files in new directories, the INI file must be edited to include the new subfiles and to augment the search path(s). For this example:

```
[RS274NGC]
...
SUBROUTINE_PATH = /home/myusername/mysubs:../../nc_files/ngcgui_lib:../../nc_files/gcmc_lib ↔
    :../../nc_files/ngcgui_lib/utilitysubs
USER_M_PATH      = /home/myusername/mymfiles:../../nc_files/ngcgui_lib/mfiles

[DISPLAY]
...
NGCGUI_SUBFILE = example.ngc
...
```

LinuxCNC (and ngcgui) use the first file found when searching directories in the search path. With this behavior, you can supersede an ngcgui_lib subfile by placing a subfile with an identical name in a directory that is found earlier in the path search. More information can be found in the INI chapter of the Integrators Manual.

5.3.4 Standalone Usage

5.3.4.1 Standalone NGCGUI

For usage, type in a terminal:

```
ngcgui --help
Usage:
  ngcgui --help | -?
  ngcgui [Options] -D nc_files_directory_name
  ngcgui [Options] -i LinuxCNC_inifile_name
  ngcgui [Options]

Options:
  [-S subroutine_file]
  [-p preamble_file]
  [-P postamble_file]
  [-o output_file]
  [-a autosend_file]      (autosend to AXIS default:auto.ngc)
  [--noauto]              (no autosend to AXIS)
  [-N | --nom2]           (no m2 terminator (use %))
  [--font [big|small|fontspec]] (default: "Helvetica -10 normal")
  [--horiz|--vert]        (default: --horiz)
  [--cwidth comment_width] (width of comment field)
  [--vwidth varname_width] (width of varname field)
  [--quiet]               (fewer comments in outfile)
  [--noiframe]            (default: frame displays image)
```

Note

As a standalone application, ngcgui handles a single subroutine file which can be invoked multiple times. Multiple standalone ngcgui applications can be started independently.

5.3.4.2 Standalone PYNGCGUI

For usage, type in a terminal:

```
pyngcgui --help
Usage:
  pyngcgui [Options] [sub_filename]
```

Options requiring values:

```

[-d | --demo] [0|1|2] (0: DEMO standalone toplevel)
                      (1: DEMO embed new notebook)
                      (2: DEMO embed within existing notebook)
[-S | --subfile      sub_filename]
[-p | --preamble     preamble_filename]
[-P | --postamble    postamble_filename]
[-i | --ini          inifile_name]
[-a | --autofile     auto_filename]
[-t | --test         testno]
[-K | --keyboardfile glade_file] (use custom popupkeyboard glade file)

```

Solo Options:

```

[-v | --verbose]
[-D | --debug]
[-N | --nom2]      (no m2 terminator (use %))
[-n | --noauto]    (save but do not automatically send result)
[-k | --keyboard]  (use default popupkeybaord)
[-s | --sendtoaxis] (send generated ngc file to AXIS GUI)

```

Notes:

A set of files is comprised of a preamble, subfile, postamble.
 The preamble and postamble are optional.
 One set of files can be specified from cmdline.
 Multiple sets of files can be specified from an inifile.
 If --ini is NOT specified:
 search for a running LinuxCNC and use its inifile

Note

As a standalone application, pyngcgui can read an INI file (or a running LinuxCNC application) to create tab pages for multiple subfiles.

5.3.5 Embedding NGCGUI

5.3.5.1 Embedding NGCGUI in AXIS

The following INI file items go in the [DISPLAY] section. (See additional sections below for additional items needed)

- *TKPKG* = *Ngcgui 1.0* - the NGCGUI package
- *TKPKG* = *Ngcguittt 1.0* - the True Type Tracer package for generating text for engraving (optional, must follow *TKPKG* = *Ngcgui*).
- *NGCGUI_FONT* = *Helvetica -12 normal* - Sets the font used
- *NGCGUI_PREAMBLE* = *in_std.ngc* - The preamble file to be added at the beginning of the subroutine. When several subroutines are concatenated, it is only added once.
- *NGCGUI_SUBFILE* = *simp.ngc* - Creates a tab from the named subroutine.
- *'NGCGUI_SUBFILE* = *""* - Creates a custom tab
- *#NGCGUI_OPTIONS* = *opt1 opt2 ...* - Ngcgui options:
 - *nonew* — Prohibits creation of new custom tab
 - *noremove* — Prohibits deleting a tab page
 - *noauto* — Do not run automatically (makeFile, then manual run)

- *noiframe* — No internal image, image on separate top level
- *TTT* = *truetype-tracer* - name of the truetype tracer program (it must be in user PATH)
- *TTT_PREAMBLE* = *in_std.ngc* - Optional, specifies filename for preamble used for ttt created sub-files. (alternate: *mm_std.ngc*)

Note

The optional truetype tracer items are used to specify an ngcgui-compatible tab page that uses the application truetype-tracer. The truetype-tracer application must be installed independently and located in the user PATH.

5.3.5.2 Embedding PYNGCGUI as a GladeVCP tab page in a GUI

The following INI file items go in the [DISPLAY] section for use with the AXIS, gscreen, or Touchy GUIs. (See additional sections below for additional items needed)

EMBED_ Items

- EMBED_TAB_NAME = *Pyngcgui* - name to appear on embedded tab
- EMBED_TAB_COMMAND = *gladevcp -x {XID} pyngcgui_axis.ui* - invokes GladeVCP
- EMBED_TAB_LOCATION = *name_of_location* - where the embedded page is located

Note

The EMBED_TAB_LOCATION specifier is not used for the AXIS GUI. While pyngcgui can be embedded in AXIS, integration is more complete when using ngcgui (using TKPKG = Ngcgui 1.0). To specify the EMBED_TAB_LOCATION for other GUIs, see the [DISPLAY Section](#) of the INI Configuration Chapter.

Note

The truetype tracer GUI front-end is not currently available for GladeVCP applications.

5.3.5.3 Additional INI File items required for ngcgui or pyngcgui

The following INI file items go in the [DISPLAY] section for any GUI that embeds either ngcgui or pyngcgui.

- *NGCGUI_FONT* = *Helvetica -12 normal* - specifies the font name,size, normal|bold
 - *NGCGUI_PREAMBLE* = *in_std.ngc* - the preamble file to be added in front of the subroutines. When concatenating several common subroutine invocations, this preamble is only added once. For mm-based machines, use *mm_std.ngc*
 - *NGCGUI_SUBFILE* = *filename1.ngc* - creates a tab from the filename1 subroutine
 - *NGCGUI_SUBFILE* = *filename2.ngc* - creates a tab from the filename2 subroutine
 - ... *etc*
 - *NGCGUI_SUBFILE* = *gcmcname1.gcmc* - creates a tab from the gcmcname1 file
 - *NGCGUI_SUBFILE* = *gcmcname2.gcmc* - creates a tab from the gcmcname2 file
-

- ... *etc*
- `NGCGUI_SUBFILE = ""` - creates a custom tab that can open any subroutine in the search path
- `NGCGUI_OPTIONS = opt1 opt2 ...` - NGCGUI options
 - *nonew* - disallow making a new custom tab
 - *noremove* - disallow removing any tab page
 - *noauto* - no autosend (use `makeFile`, then save or manually send)
 - *noiframe* - no internal image, display images on separate top level widget
 - *nom2* - do not terminate with `m2`, use `%` terminator. This option eliminates all the side effects of `m2` termination
- `GCMC_INCLUDE_PATH = dirname1:dirname2` - search directories for `gcmc` include files

This is an example of embedding NGCGUI into AXIS. The subroutines need to be in a directory specified by the `[RS274NGC]SUBROUTINE_PATH`. Some example subroutines use other subroutines so check to be sure you have the dependences, if any, in a `SUBROUTINE_PATH` directory. Some subroutines may use custom M-files which must be in a directory specified by the `[RS274NGC]USER_M_PATH`.

The G-code-meta-compiler (`gcmc`) can include statements like:

```
include("filename.inc.gcmc");
```

By default, `gcmc` includes the current directory which, for LinuxCNC, will be the directory containing the LinuxCNC INI file. Additional directories can be prepended to the `gcmc` search order with the `GCMC_INCLUDE_PATH` item.

Sample AXIS-GUI-based INI

```
[RS274NGC]
...
SUBROUTINE_PATH  = ../../nc_files/ngcgui_lib:../../ngcgui_lib/utilitysubs
USER_M_PATH      = ../../nc_files/ngcgui_lib/mfiles

[DISPLAY]
TKPKG            = Ngcgui      1.0
TKPKG            = Ngcguiittt 1.0
# Ngcgui must precede Ngcguiittt

NGCGUI_FONT      = Helvetica -12 normal
# specify filenames only, files must be in [RS274NGC]SUBROUTINE_PATH
NGCGUI_PREAMBLE  = in_std.ngc
NGCGUI_SUBFILE   = simp.ngc
NGCGUI_SUBFILE   = xyz.ngc
NGCGUI_SUBFILE   = iquad.ngc
NGCGUI_SUBFILE   = db25.ngc
NGCGUI_SUBFILE   = ihex.ngc
NGCGUI_SUBFILE   = gosper.ngc
# specify "" for a custom tab page
NGCGUI_SUBFILE   = ""
#NGCGUI_SUBFILE  = "" use when image frame is specified if
#                  opening other files is required
#                  images will be put in a top level window
NGCGUI_OPTIONS   =
#NGCGUI_OPTIONS  = opt1 opt2 ...
# opt items:
#   nonew        -- disallow making a new custom tab
#   noremove     -- disallow removing any tab page
#   noauto       -- no auto send (makeFile, then manually send)
#   noiframe     -- no internal image, image on separate top level
```

```
GCMC_INCLUDE_PATH = /home/myname/gcmc_includes

TTT                = truetype-tracer
TTT_PREAMBLE       = in_std.ngc

PROGRAM_PREFIX     = ../../nc_files
```

Note

The above is not a complete AXIS GUI INI — the items shown are those used by ngcgui. Many additional items are required by LinuxCNC to have a complete INI file.

5.3.5.4 Truetype Tracer

Ngcgui_ttt provides support for truetype-tracer (v4). It creates an AXIS tab page which allows a user to create a new ngcgui tab page after entering text and selecting a font and other parameters. (Truetype-tracer must be installed independently).

To embed ngcgui_ttt in AXIS, specify the following items in addition to ngcgui items:

```
Item:    [DISPLAY]TKPKG = Ngcgui_ttt version_number
Example: [DISPLAY]TKPKG = Ngcgui_ttt 1.0
Note:    Mandatory, specifies loading of ngcgui_ttt in an AXIS tab page
         named ttt.
         Must follow the TKPKG = Ngcgui item.

Item:    [DISPLAY]TTT = path_to_truetype-tracer
Example: [DISPLAY]TTT = truetype-tracer
Note:    Optional, if not specified, attempt to use
         /usr/local/bin/truetype-tracer.
         Specify with absolute pathname or as a simple executable
         name in which case the user PATH environment will be used
         to find the program.

Item:    [DISPLAY]TTT_PREAMBLE = preamble_filename
Example: [DISPLAY]TTT_PREAMBLE = in_std.ngc
Note:    Optional, specifies filename for preamble used for ttt
         created subfiles.
```

5.3.5.5 INI File Path Specifications

Ngcgui uses the LinuxCNC search path to find files.

The search path begins with the standard directory specified by:

```
[DISPLAY]PROGRAM_PREFIX = directory_name
```

followed by multiple directories specified by:

```
[RS274NGC]SUBROUTINE_PATH = directory1_name:directory1_name:directory3_name ...
```

Directories Directories may be specified as absolute paths or relative paths.

- Example: [DISPLAY]PROGRAM_PREFIX = /home/myname/linuxcnc/nc_files
- Example: [DISPLAY]PROGRAM_PREFIX = ~/linuxcnc/nc_files

- Example: `[DISPLAY]PROGRAM_PREFIX = ../../nc_files`

Absolute Paths An absolute path beginning with a "/" specifies a complete filesystem location. A path beginning with a "~/ " specifies a path starting from the user's home directory. A path beginning with "~username/" specifies a path starting in username's home directory.

Relative Paths Relative paths are based on the startup directory which is the directory containing the INI file. Using relative paths can facilitate relocation of configurations but requires a good understanding of linux path specifiers.

- `./d0` is the same as `d0`, e.g., a directory named `d0` in the startup directory
- `../d1` refers to a directory `d1` in the parent directory
- `../../d2` refers to a directory `d2` in the parent of the parent directory
- `../.../d3` etc.

Multiple directories can be specified with `[RS274NGC]SUBROUTINE_PATH` by separating them with colons. The following example illustrates the format for multiple directories and shows the use of relative and absolute paths.

Multiple Directories Example:

```
[RS274NGC]SUBROUTINE_PATH = ../../nc_files/ngcgui_lib:../../nc_files/ngcgui_lib/utilitysubs ↵
                           :/tmp/tmpngc'
```

This is one long line, do not continue on multiple lines. When LinuxCNC and/or ngcgui searches for files, the first file found in the search is used.

LinuxCNC (and ngcgui) must be able to find all subroutines including helper routines that are called from within ngcgui subfiles. It is convenient to place utility subs in a separate directory as indicated in the example above.

The distribution includes the `ngcgui_lib` directory and demo files for preambles, subfiles, postambles and helper files. To modify the behavior of the files, you can copy any file and place it in an earlier part of the search path. The first directory searched is `[DISPLAY]PROGRAM_PREFIX`. You can use this directory but it is better practice to create dedicated directory(ies) and put them at the beginning of the `[RS274NGC]SUBROUTINE_PATH`.

In the following example, files in `/home/myname/linuxcnc/mysubs` will be found before files in `../../nc_files/ngcgui_lib`.

Adding User Directory Example:

```
[RS274NGC]SUBROUTINE_PATH = /home/myname/linuxcnc/mysubs:../../nc_files/ngcgui_lib:../../ ↵
                           nc_files/ngcgui_lib/utilitysubs'
```

New users may inadvertently try to use files that are not structured to be compatible with ngcgui requirements. Ngcgui will likely report numerous errors if the files are not coded per its conventions. Good practice suggests that ngcgui-compatible subfiles should be placed in a directory dedicated to that purpose and that preamble, postamble, and helper files should be in separate directory(ies) to discourage attempts to use them as subfiles. Files not intended for use as subfiles can include a special comment: `"(not_a_subfile)"` so that ngcgui will reject them automatically with a relevant message.

5.3.5.6 Summary of INI File item details for NGCGUI usage

[RS274NGC]SUBROUTINE_PATH = dirname1:dirname2:dirname3 ...

Example: `[RS274NGC]SUBROUTINE_PATH = ../../nc_files/ngcgui_lib:../../nc_files/ngcgui_l`

Note: Optional, but very useful to organize subfiles and utility files.

[RS274NGC]USER_M_PATH = dirname1:dirname2:dirname3 ...

Example: [RS274NGC]USER_M_PATH = ../../nc_files/ngcgui_lib/mfiles

Note: Optional, needed to locate custom user mfiles.

[DISPLAY]EMBED_TAB_NAME = name to display on embedded tab page

Example: [DISPLAY]EMBED_TAB_NAME = Pyngcgui

Note: The entries: EMBED_TAB_NAME, EMBED_TAB_COMMAND, EMBED_TAB_LOCATION define an embedded application for several LinuxCNC GUIs.

[DISPLAY]EMBED_TAB_COMMAND = programname followed by arguments

Example: [DISPLAY]EMBED_TAB_COMMAND = gladevcp -x {XID} pyngcgui_axis.ui

Note: For GladeVCP applications, see the [GladeVCP Chapter](#).

[DISPLAY]EMBED_TAB_LOCATION = name_of_location

Example: [DISPLAY]EMBED_TAB_LOCATION = notebook_main

Note: See example INI files for possible locations.

Not required for the AXIS GUI.

[DISPLAY]PROGRAM_PREFIX = dirname

Example: [DISPLAY]PROGRAM_PREFIX = ../../nc_files

Note: Mandatory and needed for numerous LinuxCNC functions.

It is the first directory used in the search for files.

[DISPLAY]TKPKG = Ngcgui version_number

Example: [DISPLAY]TKPKG = Ngcgui 1.0

Note: Required only for AXIS GUI embedding.

Specifies loading of NGCGUI AXIS tab pages.

[DISPLAY]NGCGUI_FONT = font_descriptor

Example: [DISPLAY]NGCGUI_FONT = Helvetica -12 normal

Note: Optional, font_descriptor is a tcl-compatible font specifier with items for fonttype -fontsize fontweight.

Default is: Helvetica -10 normal.

Smaller font sizes may be useful for small screens.

Larger font sizes may be helpful for touch screen applications .

[DISPLAY]NGCGUI_SUBFILE = subfile_filename

Example: [DISPLAY]NGCGUI_SUBFILE = simp.ngc

Example: [DISPLAY]NGCGUI_SUBFILE = square.gcmc

Example: [DISPLAY]NGCGUI_SUBFILE = ""

Note: Use one or more items to specify ngcgui-compatible subfiles or gcmc programs that require a tab page on startup.

A "Custom" tab will be created when the filename is "".

A user can use a "Custom" tab to browse the file system and identify preamble, subfile, and postamble files.

[DISPLAY]NGCGUI_PREAMBLE = preamble_filename

Example: [DISPLAY]NGCGUI_PREAMBLE = in_std.ngc

Note: Optional, when specified, the file is prepended to a subfile.

Files created with "Custom" tab pages use the preamble specified with the page.

[DISPLAY]NGCGUI_POSTAMBLE = postamble_filename

Example: [DISPLAY]NGCGUI_POSTAMBLE = bye.ngc

Note: Optional, when specified, the file is appended to a subfiles.

Files created with "Custom" tab pages use the postamble specified with the page.

[DISPLAY]NGCGUI_OPTIONS = opt1 opt2 ...

Example: [DISPLAY]NGCGUI_OPTIONS = nonew noremove

Note: Multiple options are separated by blanks.

By default, ngcgui configures tab pages so that:

- 1) a user can make new tabs;
- 2) a user can remove tabs (except for the last remaining one);
- 3) finalized files are automatically sent to LinuxCNC;
- 4) an image frame (iframe) is made available to display an image for the subfile (if an image is provided);
- 5) the ngcgui result file sent to LinuxCNC is terminated with an m2 (and incurs m2 side-effects).

The options `nonew`, `noremove`, `noauto`, `noiframe`, `nom2` respectively disable these default behaviors.

By default, if an image (`.png`, `.gif`, `.jpg`, `.pgm`) file is found in the same directory as the subfile, the image is displayed in the iframe. Specifying the `noiframe` option makes available additional buttons for selecting a preamble, subfile, and postamble and additional checkboxes. Selections of the checkboxes are always available with special keys:

Ctrl-R Toggle "Retain values on Subfile read",

Ctrl-E Toggle "Expand subroutine",

Ctrl-a Toggle "Autosend",

Ctrl-k lists all keys and functions.

If `noiframe` is specified and an image file is found, the image is displayed in a separate window and all functions are available on the tab page.

The `NGCGUI_OPTIONS` apply to all `ngcgui` tabs except that the `nonew`, `noremove`, and `noiframe` options are not applicable for "Custom" tabs. Do not use "Custom" tabs if you want to limit the user's ability to select subfiles or create additional tab pages.

[DISPLAY]GCMC_INCLUDE_PATH = dirname1:dirname2:...

Example: `[DISPLAY]GCMC_INCLUDE_PATH = /home/myname/gcmc_includes:/home/myname/gcmc_inc`

Note: Optional, each directory will be included when `gcmc` is invoked using the option: `--include dirname`.

5.3.6 File Requirements for NGCGUI Compatibility

5.3.6.1 Single-File Gcode (.ngc) Subroutine Requirements

An NGCGUI-compatible subfile contains a single subroutine definition. The name of the subroutine must be the same as the filename (not including the `.ngc` suffix). LinuxCNC supports named or numbered subroutines, but only named subroutines are compatible with NGCGUI. For more information see the [O-Codes](#) chapter.

The first non-comment line should be a `sub` statement.

The last non-comment line should be an `endsub` statement.

examp.ngc:

```
(info: info_text_to_appear_at_top_of_tab_page)
; comment line beginning with semicolon
( comment line using parentheses)
o<examp> sub
  BODY_OF_SUBROUTINE
o<examp> endsub
; comment line beginning with semicolon
( comment line using parentheses)
```

The body of the subroutine should begin with a set of statements that define local named parameters for each positional parameter expected for the subroutine call. These definitions must be consecutive beginning with `#1` and ending with the last used parameter number. Definitions must be provided for each of these parameters (no omissions).

Parameter Numbering

```
#<xparm> = #1
#<yparm> = #2
#<zparm> = #3
```

LinuxCNC considers all numbered parameters in the range #1 thru #30 to be calling parameters so ngcgui provides entry boxes for any occurrence of parameters in this range. It is good practice to avoid use of numbered parameters #1 through #30 anywhere else in the subroutine. Using local, named parameters is recommended for all internal variables.

Each defining statement may optionally include a special comment and a default value for the parameter.

Statement Prototype

```
#<vname> = #n (=default_value)
or
#<vname> = #n (comment_text)
or
#<vname> = #n (=default_value comment_text)
```

Parameter Examples

```
#<xparm> = #1 (=0.0)
#<yparm> = #2 (Ystart)
#<zparm> = #3 (=0.0 Z start setting)
```

If a default_value is provided, it will be entered in the entry box for the parameter on startup.

If comment_text is included, it will be used to identify the input instead of the parameter name.

Global Named Parameters

Notes on global named parameters and ngcgui:

(global named parameters have a leading underscore in the name, like #<_someglobalname>)

As in many programming languages, use of globals is powerful but can often lead to unexpected consequences. In LinuxCNC, existing global named parameters will be valid at subroutine execution and subroutines can modify or create global named parameters.

Passing information to subroutines using global named parameters is discouraged since such usage requires the establishment and maintenance of a well-defined global context that is difficult to maintain. Using numbered parameters #1 thru #30 as subroutine inputs should be sufficient to satisfy a wide range of design requirements. Ngcgui supports some input global named parameter but their usage is obsolete and not documented here.

While input global named parameters are discouraged, LinuxCNC subroutines must use global named parameters for returning results. Since ngcgui-compatible subfiles are aimed at GUI usage, return values are not a common requirement. However, ngcgui is useful as a testing tool for subroutines which do return global named parameters and it is common for ngcgui-compatible subfiles to call utility subroutine files that return results with global named parameters.

To support these usages, ngcgui ignores global named parameters that include a colon (:) character in their name. Use of the colon (:) in the name prevents ngcgui from making entryboxes for these parameters.

Global Named Parameters Example

```
o<examp> sub
...
#<_examp:result> = #5410      (return the current tool diameter)
...
o<helper> call [#<x1>] [#<x2>] (call a subroutine)
#<xresult> = #<_helper:answer> (immediately localize the helper global result)
#<_helper:answer> = 0.0      (nullify global named parameter used by subroutine)
...
o<examp> endsb
```


In the above example, the utility subroutine will be found in a separate file named `helper.ngc`. The helper routine returns a result in a global named parameter named `#<_helper:answer`.

For good practice, the calling subfile immediately localizes the result for use elsewhere in the subfile and the global named parameter used for returning the result is nullified in an attempt to mitigate its inadvertent use elsewhere in the global context. (A nullification value of 0.0 may not always be a good choice).

Ngcgui supports the creation and concatenation of multiple features for a subfile and for multiple subfiles. It is sometimes useful for subfiles to determine their order at runtime so ngcgui inserts a special global parameter that can be tested within subroutines. The parameter is named `#<_feature:>`. Its value begins with a value of 0 and is incremented for each added feature.

Additional Features A special *info* comment can be included anywhere in an ngcgui-compatible subfile. The format is:

```
(info: info_text)
```

The `info_text` is displayed near the top of the ngcgui tab page in AXIS.

Files not intended for use as subfiles can include a special comment so that ngcgui will reject them automatically with a relevant message.

```
(not_a_subfile)
```

An optional image file (`.png`, `.gif`, `.jpg`, `.pgm`) can accompany a subfile. The image file can help clarify the parameters used by the subfile. The image file should be in the same directory as the subfile and have the same name with an appropriate image suffix, e.g. the subfile `example.ngc` could be accompanied by an image file `examp.png`. Ngcgui attempts to resize large images by subsampling to a size with maximum width of 320 and maximum height of 240 pixels.

None of the conventions required for making an ngcgui-compatible subfile preclude its use as general purpose subroutine file for LinuxCNC.

The LinuxCNC distribution includes a library (`ngcgui_lib` directory) that includes both example ngcgui-compatible subfiles and utility files to illustrate the features of LinuxCNC subroutines and ngcgui usage. Another library (`gcmc_lib`) provides examples for subroutine files for the G-code meta compiler (`gcmc`).

Additional user submitted subroutines can be found on the Forum in the Subroutines Section.

5.3.6.2 Gcode-meta-compiler (.gcmc) file requirements

Files for the Gcode-meta-compiler (`gcmc`) are read by ngcgui and it creates entry boxes for variables tagged in the file. When a feature for the file is finalized, ngcgui passes the file as input to the `gcmc` compiler and, if the compile is successful, the resulting G-code file is sent to LinuxCNC for execution. The resulting file is formatted as single-file subroutine; `.gcmc` files and `.ngc` files can be intermixed by ngcgui.

The variables identified for inclusion in ngcgui are tagged with lines that will appear as comments to the `gcmc` compiler.

Variable Tags Formats

```
//ngcgui: varname1 =  
//ngcgui: varname2 = value2  
//ngcgui: varname3 = value3, label3;
```

Variable Tags Examples

```
//ngcgui: zsafe =  
//ngcgui: feedrate = 10  
//ngcgui: xl = 0, x limit
```

For these examples, the entry box for varname1 will have no default, the entry box for varname2 will have a default of value2, and the entry box for varname 3 will have a default of value 3 and a label label3 (instead of varname3). The default values must be numbers.

To make it easier to modify valid lines in a gcmc file, alternate tag line formats accepted. The alternate formats ignore trailing semicolons (;) and trailing comment markers (//) With this provision, it is often makes it possible to just add the //ngcgui: tag to existing lines in a .gcmc file.

Alternate Variable Tags Formats

```
//ngcgui: varname2 = value2;
//ngcgui: varname3 = value3; //, label3;
```

Alternate Variable Tags Examples

```
//ngcgui: feedrate = 10;
//ngcgui: xl = 0; //, x limit
```

An info line that will appear at the top of a tab page may be optionally included with a line tagged as:

Info tag

```
//ngcgui: info: text_to_appear_at_top_of_tab_page
```

When required, options can be passed to the gcmc compiler with a line tagged:

Option line tag format

```
//ngcgui: -option_name [ [=] option_value]
```

Option line tag Examples

```
//ngcgui: -I
//ngcgui: --imperial
//ngcgui: --precision 5
//ngcgui: --precision=6
```

Options for gcmc are available with the terminal command:

```
gcmc --help
```

A gcmc program by default uses metric mode. The mode can be set to inches with the option setting:

```
//ngcgui: --imperial
```

A preamble file, if used, can set a mode (g20 or g21) that conflicts with the mode used by a gcmc file. To ensure that the gcmc program mode is in effect, include the following statement in the .gcmc file:

```
include("ensure_mode.gcmc")
```

and provide a proper path for gcmc include_files in the INI file, for example:

```
[DISPLAY]
GCMC_INCLUDE_PATH = ../../nc_files/gcmc_lib
```

5.3.7 DB25 Example

The following shows the DB25 subroutine.

In the first photo you see where you fill in the blanks for each variable.



This photo shows the backplot of the DB25 subroutine.



This photo shows the use of the new button and the custom tab to create three DB25 cutouts in one program.



5.3.8 Creating a subroutine

- For creating a subroutine for use with Ngcgui, the filename and the subroutine name must be the same.
- The file must be placed in the subdirectory pointed to in the INI file.
- On the first line there may be a comment of type info:
- The subroutine must be surrounded by the sub and endsub tags.
- The variables used must be numbered variables and must not skip number.
- Comments and presets may be included.

Subroutine Skeleton Example

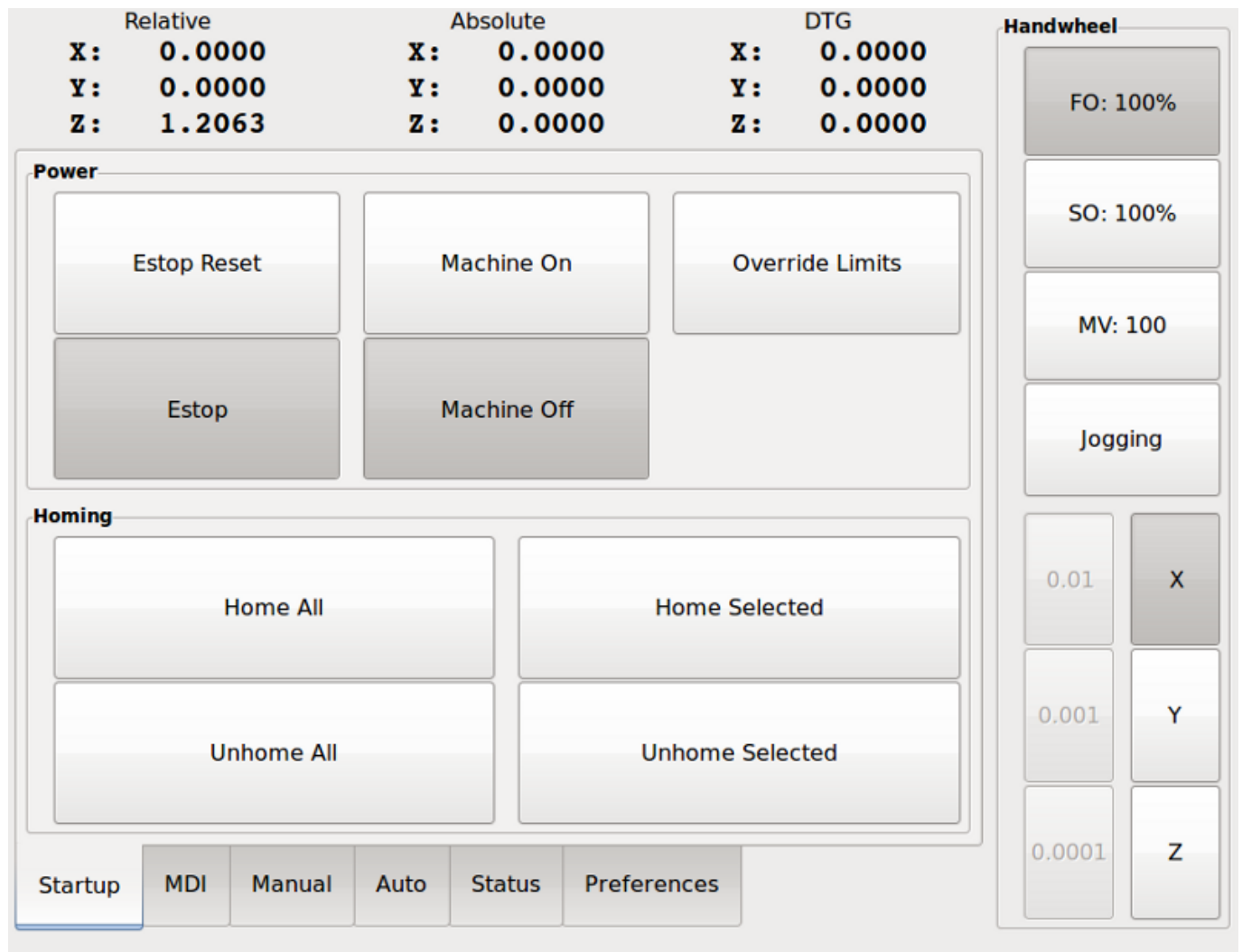
```
(info: simp -- simple exemple de sous-programme -- Ctrl-U pour éditer)
o<simp> sub
  #<ra>          = #1 (=0.6 Rayon A) ;Exemple de paramètre avec un commentaire
  #<radius_b>    = #2 (=0.4)          ;Exemple de paramètre sans commentaire
  #<feedrate>    = #3 (Feedrate)      ;Exemple de paramètre sans preset
  g0x0y0z1
  g3 i#<ra> f#<feedrate>
  g3 i[0-#<radius_b>]
o<simp> endsub
```

5.4 The Touchy Graphical User Interface

Touchy is a user interface for LinuxCNC meant for use on machine control panels, and therefore does not require keyboard or mouse.

It is meant to be used with a touch screen, and works in combination with a wheel/MPG and a few buttons and switches.

The *Handwheel* tab has radio buttons to select between *Feed Override*, *Spindle Override*, *Maximum Velocity* and *Jogging* functions for the wheel/MPG input. Radio buttons for axis selection and increment for jogging are also provided.



5.4.1 Panel Configuration

5.4.1.1 HAL connections

Touchy looks in the INI file, under the heading *[HAL]* for entries of *POSTGUI_HALFILE=<filename>*. Typically *<filename>* would be *touchy_postgui.hal*, but can be any legal filename. These commands are executed after the screen is built, guaranteeing the widget HAL pins are available. You can have multiple line of *POSTGUI_HALFILE=<filename>* in the INI. Each will be run one after the other in the order they appear in the INI file.

Note

Touchy used to require that you create a file named *touchy.hal* in your configuration directory (the directory your INI file is in). For legacy reasons this will continue to work, but INI based postgui files are preferred.

For more information on HAL files and the net command see the [HAL Basics](#).

Touchy has several output pins that are meant to be connected to the motion controller to control wheel jogging:

- *touchy.jog.wheel.increment*, which is to be connected to the *axis.N.jog-scale* pin of each axis N.
- *touchy.jog.wheel.N*, which is to be connected to *axis.N.jog-enable* for each axis N.

Note

N represents the axis number 0-8.

- In addition to being connected to *touchy.wheel-counts*, the wheel counts should also be connected to *axis.N.jog-counts* for each axis N. If you use HAL component *ilowpass* to smooth wheel jogging, be sure to smooth only *axis.N.jog-counts* and not *touchy.wheel-counts*.

Required controls

- Abort button (momentary contact) connected to the HAL pin *touchy.abort*
- Cycle start button (momentary contact) connected to *touchy.cycle-start*
- Wheel/MPG, connected to *touchy.wheel-counts* and motion pins as described above
- Single block (toggle switch) connected to *touchy.single-block*

Optional controls

- For continuous jog, one center-off bidirectional momentary toggle (or two momentary buttons) for each axis, hooked to *touchy.jog.continuous.x.negative*, *touchy.jog.continuous.x.positive*, etc.
- If a quill up button is wanted (to jog Z to the top of travel at top speed), a momentary button connected to *touchy.quill-up*.

Optional panel lamps

- *touchy.jog.active* shows when the panel jogging controls are live
- *touchy.status-indicator* is on when the machine is executing G-code, and flashes when the machine is executing but is in pause/feedhold.

5.4.1.2 Recommended for any setup

- Estop button hardwired in the estop chain

5.4.2 Setup

5.4.2.1 Enabling Touchy

To use Touchy, in the *[DISPLAY]* section of your ini file change the display selector line to *DISPLAY = touchy*

5.4.2.2 Preferences

When you start Touchy the first time, check the Preferences tab. If using a touchscreen, choose the option to hide the pointer for best results.

The Status Window is a fixed height, set by the size of a fixed font. This can be affected by the Gnome DPI, configured in System / Preferences / Appearance / Fonts / Details. If the bottom of the screen is cut off, reduce the DPI setting.

All other font sizes can be changed on the Preferences tab.

5.4.2.3 Macros

Touchy can invoke O-word macros using the MDI interface. To configure this, in the *[TOUCHY]* section of the ini file, add one or more *MACRO* lines. Each should be of the format

```
MACRO=increment xinc yinc
```

In this example, increment is the name of the macro, and it accepts two parameters, named xinc and yinc.

Now, place the macro in a file named *increment.ngc*, in the *PROGRAM_PREFIX* directory or any directory in the *SUBROUTINE_PATH*.

It should look like:

```
O<increment> sub
G91 G0 X#1 Y#2
G90
O<increment> endsub
```

Notice the name of the sub matches the file name and macro name exactly, including case.

When you invoke the macro by pressing the Macro button on the MDI tab in Touchy, you can enter values for xinc and yinc. These are passed to the macro as *#1* and *#2* respectively. Parameters you leave empty are passed as value 0.

If there are several different macros, press the Macro button repeatedly to cycle through them.

In this simple example, if you enter -1 for xinc and press cycle start, a rapid *G0* move will be invoked, moving one unit to the left.

This macro capability is useful for edge/hole probing and other setup tasks, as well as perhaps hole milling or other simple operations that can be done from the panel without requiring specially-written gcode programs.

5.5 Gscreen

5.5.1 Introduction

Gscreen is an infrastructure to display a custom screen to control LinuxCNC. Gscreen borrows heavily from GladeVCP. GladeVCP uses the GTK widget editor GLADE to build virtual control panels (VCP) by point and click. Gscreen combines this with python programming to create a GUI screen for running a CNC machine.

Gscreen is customizable if you want different buttons and status LEDs. Gscreen supports GladeVCP which is used to add controls and indicators. To customize Gscreen you use the Glade editor. Gscreen is not restricted to adding a custom panel on the right or a custom tab it is fully editable.

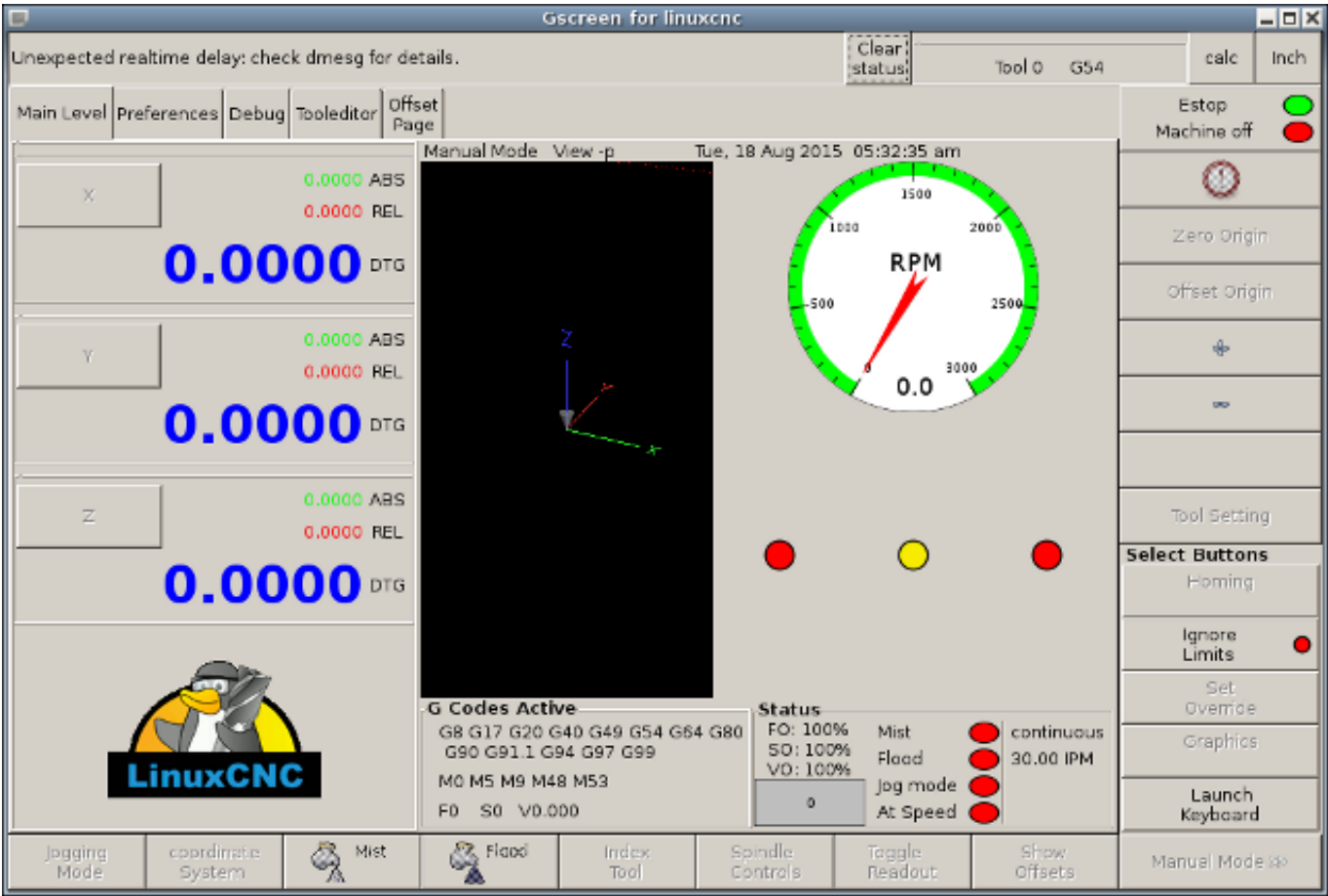


Figure 5.23: Gscreen Default Screen

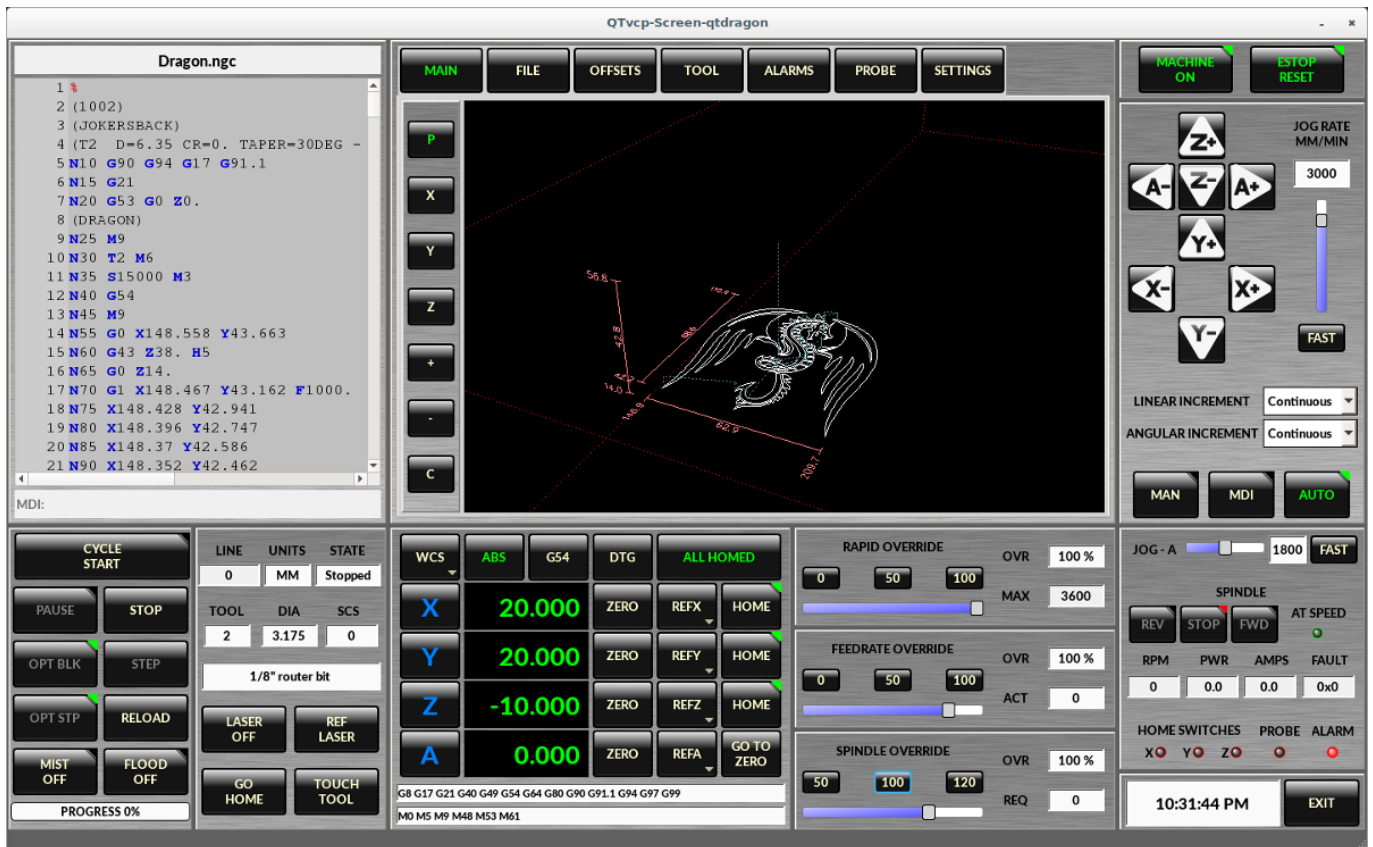


Figure 5.24: Gscreen Silverdragon Screen

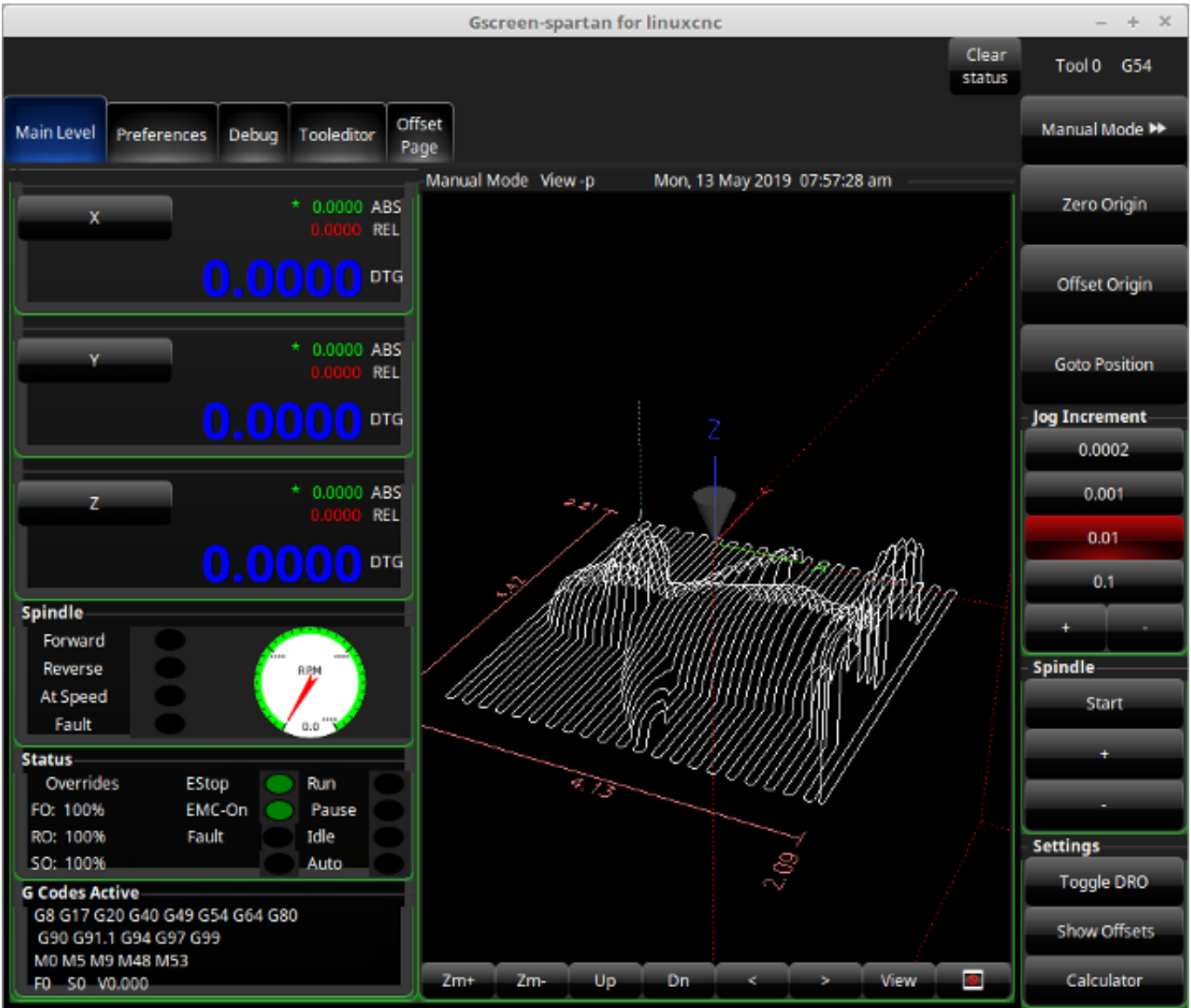


Figure 5.25: Gscreen Spartan Screen

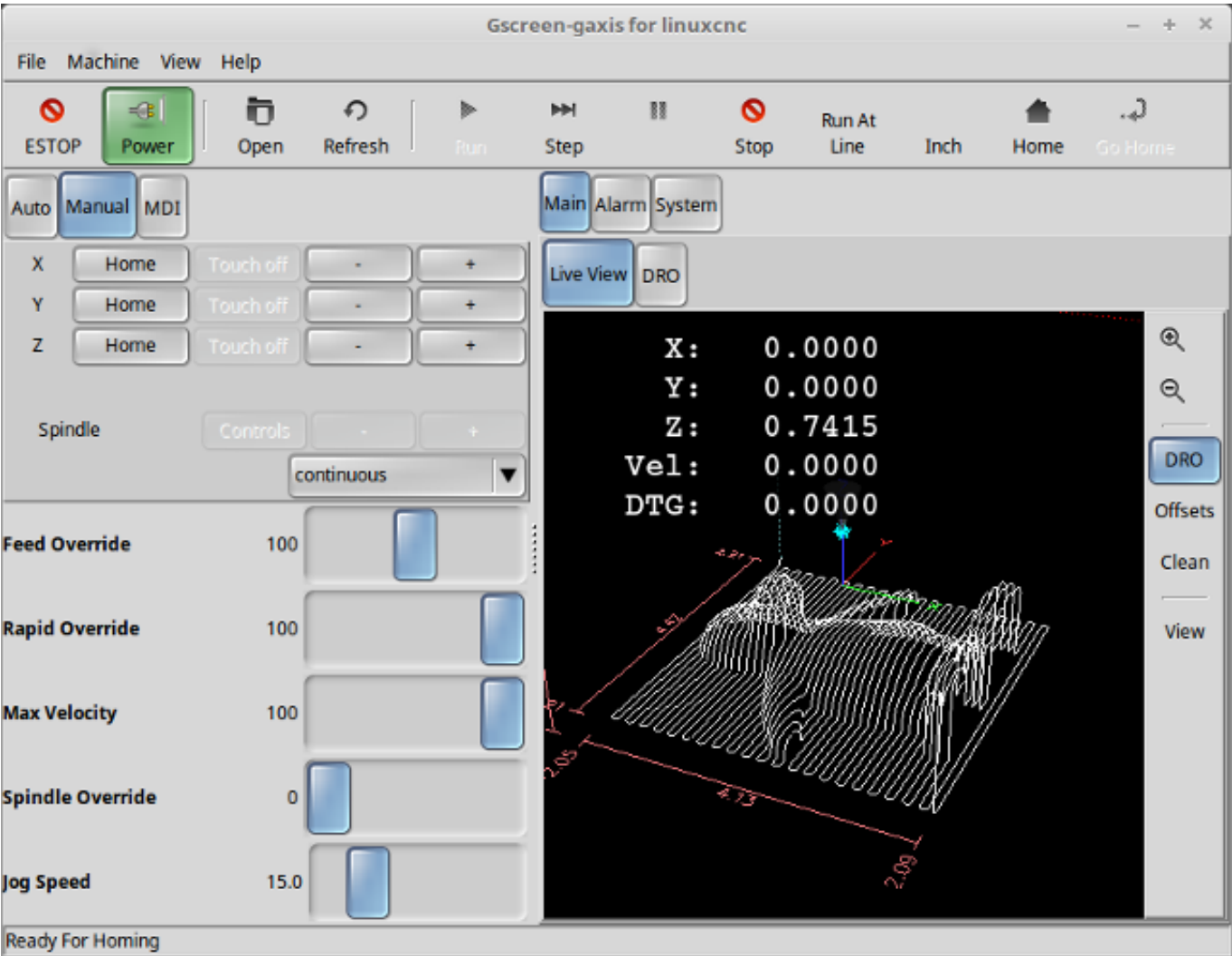


Figure 5.26: Gscreen Gaxis Screen



Figure 5.27: Gscreen Industrial Screen

Gscreen is based on *Glade* (the editor), *PyGTK* (the widget toolkit), and *GladeVCP* (LinuxCNC's connection to Glade and PyGTK). GladeVCP has some special widgets and actions added just for LinuxCNC. A widget is just the generic name used for the buttons, sliders, labels etc of the PyGTK toolkit.

5.5.1.1 Glade File

A Glade file is a text file organized in the XML standard that describes the layout and the widgets of the screen. PyGTK uses this file to actually display and react to those widgets. The Glade editor makes it relatively easy to build and edit this file. You must use the Glade 3.38.2 editor that uses the GTK3 widgets.

5.5.1.2 PyGTK

PyGTK is the python binding to GTK. GTK is the *toolkit* of visual widgets, it's programmed in C. PyGTK uses Python to *bind* with GTK.

5.5.2 GladeVCP

[GladeVCP](#) binds LinuxCNC, HAL, PyGTK and Glade all together. LinuxCNC requires some special widgets so GladeVCP supplies them. Many are just HAL extensions to existing PyGTK widgets. GladeVCP creates the HAL pins for the special widgets described in the Glade file. GladeVCP also allows one to add python commands to interact with the widgets, to make them do things not available in their default form. If you can build a GladeVCP panel you can customize Gscreen!

5.5.2.1 Overview

There are two files that can be used, individually or in combination to add customizations. Local glade files and handler files. Normally Gscreen uses the stock Glade file and possibly a handler file (if using a sample *skin*). You can specify Gscreen to use *local* Glade and handler files. Gscreen looks in the folder that holds all the configuration files for the configuration you selected.

Local Glade Files If present, local glade files in the configuration folder will be loaded instead of the stock Glade files. Local Glade files allow you to use your customized designs rather than the default screens. There is a switch in the INI file to set the base name: `-c name` so Gscreen looks for `MYNAME.glade` and `MYNAME_handler.py`.

You can tell Gscreen to just load the Glade file and not connect its internal signals to it. This allows gscreen to load any GTK builder saved Glade file. This means you can display a completely custom screen, but also requires you to use a handler file. Gscreen uses the Glade file to define the widgets, so it can show and interact with them. Many of them have specific names, others have Glade given generic names. If the widget will be displayed but never changed then a generic name is fine. If one needs to control or interact with the widget then a hopefully purposeful name is given (all names must be unique). Widgets can also have signals defined for them in the GLADE editor. It defines what signal is given and what method to call.

Modifying Stock Skins If you change the name of a widget, Gscreen might not be able to find it. If this widget is referenced to from python code, at best this makes the widget not work anymore at worst it will cause an error when loading Gscreen's default screens don't use many signals defined in the editor, it defines them in the python code. If you move (cut and paste) a widget with signals, the signals will not be copied. You must add them again manually.

Handler Files A handler file is a file containing python code, which Gscreen adds to its default routines. A handler file allows one to modify defaults, or add logic to a Gscreen skin without having to modify Gscreen proper. You can combine new functions with Gscreen's function to modify behavior as you like. You can completely bypass all of Gscreen's functions and make it work completely differently. If present a handler file named `gscreen_handler.py` (or `MYNAME_handler.py` if using the INI switch) will be loaded and registered only one file is allowed Gscreen looks for the handler file, if found it will look for specific function names and call them instead of the default ones. If adding widgets you can set up signal calls from the Glade editor to call routines you have written in the handler file. In this way you can have custom behavior. Handler routines can call Gscreen's default routines, either before or after running its own code. In this way you can tack on extra behavior such as adding a sound. Please see the [GladeVCP Chapter](#) for the basics to GladeVCP handler files. Gscreen uses a very similar technique.

Themes Gscreen uses the PyGTK toolkit to display the screen. Pygtk is the Python language binding to GTK. GTK supports *themes*. Themes are a way to modify the look and feel of the widgets on the screen. For instance the color or size of buttons and sliders can be changed using themes. There are many GTK themes available on the web. Themes can also be customized to modify visuals of particular named widgets. This ties the theme file to the glade file more tightly. Some of the sample screen skins allow the user to select any of the themes on the system. The sample *gscreen* is an example. Some will load the theme that is the same name in the config file. The sample *gscreen-gaxis* is an example. This is done by putting the theme folder in the config folder that has the INI and HAL files and naming it: `SCREENNAME_theme` (SCREENNAME being the base name of the files eg. *gaxis* theme) Inside this folder is another folder call `gtk-2.0`, inside that is the theme files. If you add this file, Gscreen

will default to this theme on start up. gscreen-gaxis has a sample custom theme that looks for certain named widgets and changes the visual behavior of those specific widgets. The Estop and machine-on buttons use different colors then the rest of the buttons so that they stand out. This is done in the handler file by giving them specific names and by adding specific commands in the theme's gtkrc file. For some info on GTK theming (The sample theme uses the pixmap theme engine) See:

[GTK Themes](#)

[Pixmap Theme Engine](#)

5.5.2.2 Build a GladeVCP Panel

Gscreen is just a big complicated GladeVCP panel, with python code to control it. To customize it we need the Glade file loaded in the Glade editor.

Installed LinuxCNC If you have LinuxCNC 2.6+ installed on Ubuntu 10.04 just start the Glade editor from the applications menu or from the terminal. Newer versions of Linux will require you to install Glade 3.8.0 - 3.8.6 (you may need to compile it yourself).

RIP compiled commands Using a compiled from source version of [LinuxCNC](#) open a terminal and `cd` to the top of the LinuxCNC folder. Set up the environment by entering `./scripts/rip-environment` now enter `glade`, you see a bunch of warnings in the terminal that you can ignore and the editor should open. The stock Gscreen Glade file is in: `src/emc/usr_intf/gscreen/` sample skins are in `/share/gscreen/skins/`. This should be copied to a configuration folder. Or you can make a clean-sheet Glade file by saving it in a configuration folder.

Ok you have loaded the stock Glade file and now can edit it. The first thing you notice is it does not look in the editor like what it's displayed like Gscreen uses some tricks, such as hiding all boxes of buttons except one and changing that one depending on the mode. The same goes for notebooks, some screens use notebooks with the tabs not shown. To change pages in the editor you need to temporarily show those tabs.

When making changes it is far easier to add widgets then subtract widgets and still have the screen work properly making objects *not visible* is one way to change the display without getting errors. This won't always work some widgets will be set visible again. Changing the names of Gscreen's regular widgets is probably not gonna work well without changing the python code, but moving a widget while keeping the name is usually workable.

Gscreen leverages GladeVCP widgets as much as possible, to avoid adding python code. Learning about [GladeVCP](#) widgets is a prerequisite. If the existing widgets give you the function you want or need then no python code needs be added, just save the Glade file in your configuration folder. If you need something more custom then you must do some python programming. The name of the parent window needs to be `window1`. Gscreen assumes this name.

Remember, if you use a custom screen option YOU are responsible for fixing it (if required) when updating LinuxCNC.

5.5.3 Building a simple clean-sheet custom screen

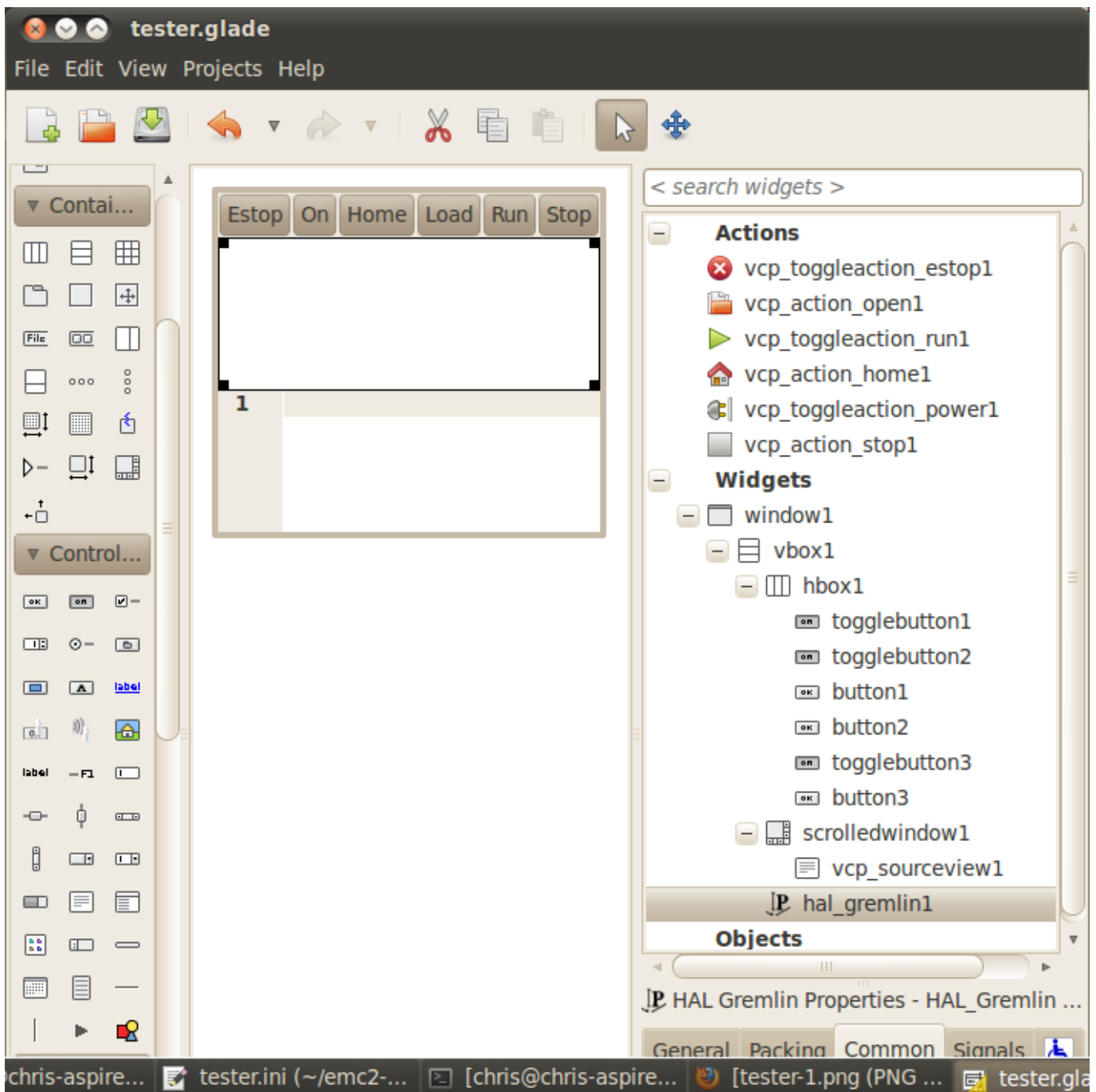


Lets build a simple usable screen. Build this in the Glade editor (if using a RIP package run it from a terminal after using `. scripts/rip-environment`).

Things to note:

- The top level window must be called the default name, *window1* - Gscreen relies on this.
- Add actions by right clicking, and selecting *add as toplevel widget* they don't add anything visual to the window but are added to the right most action list. Add all the ones you see on the top right.
- After adding the actions we must link the buttons to the actions for them to work (see below).
- The gremlin widget doesn't have a default size so setting a requested size is helpful (see below).
- The sourceview widget will try to use the whole window so adding it to a scrolled window will cover this (This is already been done in the example).
- The buttons will expand as the window is made larger which is ugly so we will set the box they are in, to not expand (see below).
- The button types to use depend on the VCP_action used -eg `vcp_toggle_action` usually require toggle buttons (Follow the example for now).

- The buttons in this example are regular buttons not HAL buttons. We don't need the HAL pins.



In this screen we are using VCP actions to communicate to LinuxCNC the actions we want. This allows us standard functions without adding python code in the handler file. Let's link the estop toggle button to the estop action. Select the estop toggle button and under the general tab look for *Related Action* and click the button beside it. Now select the toggle estop action. Now the button will toggle estop on and off when clicked. Under the general tab you can change the text of the button's label to describe it's function. Do this for all the buttons.

Select the gremlin widget click the common tab and set the requested height to 100 and click the checkbox beside it.

Click the horizontal box that holds the buttons. Click the packing tab and click *expand* to *No*.

Save it as `tester.glade` and save it in `sim/gscreen/gscreen_custom/` folder. Now launch LinuxCNC and click to `sim/gscreen/gscreen_custom/tester` and start it. If all goes well our screen will pop up and the buttons will do their job. This works because the `tester.ini` tells gscreen to look for and load `tester.glade` and `tester_handler.py`. The `tester_handler.py` file is included in that folder and is coded just show the screen and not much else. Since the special widgets directly communicate with LinuxCNC you can still do useful things. If your screen needs are covered by the available special widgets then this is as far as you need to go to build a screen. If you want something more there are still many tricks available from just adding *function calls* to get canned behaviour. To coding your own python code to customize exactly what you want. But that means learning about handler files.

5.5.4 Handler file example

There are special functions Gscreen checks the handler file for. If you add these in you handler file Gscreen will call them instead of gscreen's internal same-named functions.

- `initialize_preferences(self)`: You can install new preference routines.
- `initialize_keybindings(self)` You can install new keybinding routines. In most cases you won't want to do this, you will want to override the individual keybinding calls. You can also add more keybindings that will call an arbitrary function.
- `initialize_pins(self)`: makes / initializes HAL pins
- `connect_signals(self,handlers)`: If you are using a completely different screen the default Gscreen you must add this or gscreen will try to connect signals to widgets that are not there. Gscreen's default function is called with `self.gscreen.connect_signals(handlers)` If you wish to just add extra signals to your screen but still want the default ones call this first then add more signals. If you signals are simple (no user data passed) then you can also use the Glade signal selection in the Glade editor.
- `initialize_widgets(self)`: You can use this to set up any widgets. Gscreen usually calls `self.gscreen.initialize_widgets()` which actually calls many separate functions. If you wish to incorporate some of those widgets then just call those functions directly. or add `self.gscreen.init_show_windows()` so widgets are just shown. Then if desired, initialize/adjust your new widgets.
- `initialize_manual_toolchange(self)`: allows a complete revamp of the manual toolchange system.
- `set_restart_line(self.line)`:
- `timer_interrupt(self)`: allows one to complete redefine the interrupt routine This is used for calling `periodic()` and checking for errors from `linuxcnc.status`.
- `check_mode(self)`: used to check what mode the screen is in. Returns a list[] 0 -manual 1- mdi 2- auto 3- jog.
- `on_tool_change(self,widget)`: You can use this to override the manual tool change dialog -this is called when `gscreen.tool-change` changes state.
- `dialog_return(self,dialog_widget,displaytype,pinname)`: Use this to override any user message or manual tool change dialog. Called when the dialog is closed.
- `periodic(self)`: This is called every (default 100) milliseconds. Use it to update your widgets/HAL pins. You can call Gscreen regular `periodic` afterwards too, `self.gscreen.update_position()` or just add pass to not update anything. Gscreen's `update_position()` actually calls many separate functions. If you wish to incorporate some of those widgets then just call those functions directly.

You can also add you own functions to be called in this file. Usually you would add a signal to a widget to call your function.

5.5.4.1 Adding Keybindings Functions

Our tester example would be more useful if it responded to keyboard commands. There is a function called `keybindings()` that tries to set this up. While you can override it completely, we didn't - but it assumes some things. It assumes the estop toggle button is called *button_estop* and that F1 key controls it. It assumes the power button is called *button_machine_on* and the F2 key controls it. These are easily fixed by renaming the buttons in the Glade editor to match. But instead we are going to override the standard calls and add our own.

Add these command to the handler file:

```
# override Gscreen Functions
# keybinding calls
def on_keycall_ESTOP(self,state,SHIFT,CNTRL,ALT):
    if state: # only if pressed, not released
        self.widgets.togglebutton1.emit('activate')
        self.gscreen.audio.set_sound(self.data.alert_sound)
        self.gscreen.audio.run()
        return True # stop progression of signal to other widgets
def on_keycall_POWER(self,state,SHIFT,CNTRL,ALT):
    if state:
        self.widgets.togglebutton2.emit('activate')
        return True
def on_keycall_ABORT(self,state,SHIFT,CNTRL,ALT):
    if state:
        self.widgets.button3.emit('activate')
        return True
```

So now we have overridden Gscreen's function calls of the same name and deal with them in our handler file. We now reference the widgets by the name we used in the Glade editor. We also added a built in gscreen function to make a sound when Estop changes. Note that we we call Gscreen's built in functions we must use `self.gscreen.[FUNCTION NAME]()`. If we used `self.[FUNCTION NAME]()` it would call the function in our handler file.

Lets add another key binding that loads halmeter when F4 is pressed.

In the handler file under *def initialize_widgets(self):* change to:

```
def initialize_widgets(self):
    self.gscreen.init_show_windows()
    self.gscreen.keylookup.add_conversion('F4','TEST','on_keycall_HALMETER')
```

Then add these functions under the *HandlerClass* class:

```
def on_keycall_HALMETER(self,state,SHIFT,CNTRL,ALT):
    if state:
        self.gscreen.on_halmeter()
        return True
```

This adds a keybinding conversion that directs gscreen to call `on_keycall_HALMETER` when F4 is pressed.

Then we add the function to the handle file to call a Gscreen builtin function to start halmeter.

5.5.4.2 Linuxcnc State Status

The module *Gstat* polls linuxcnc's state every 100ms and sends callback messages to user functions when state changes. You can register messages to act on specific state changes. As an example we will register to get *file-loaded* messages when linuxcnc loads a new file. First we must import the module and instantiate it: In the import section of the handler file add:

```
from hal_glib import GStat
GSTAT = GStat()
```

In the handler file under `def __init__(self):` add:

```
GSTAT.connect('file-loaded', self.update_filepath)
```

Then in the *HandlerClass*, add the function:

```
self.update_filepath(self, obj, path):
    self.widgets.my_path_label.set_text(path)
```

When linuxcnc loads a new file, Gstat will send a callback message to the function *update_filepath*. In this example we update a label with that path name (assuming there is a label named *my_path_label*) in the GLADE file.

5.5.4.3 Jogging Keys

There are no special widgets to do screen-button jogging, so we must do it with python code. Under the `connect_signals` function add this code:

```
for i in('x','y','z'):
    self.widgets[i+'neg'].connect("pressed", self['jog_'+i],0,True)
    self.widgets[i+'neg'].connect("released", self['jog_'+i],0,False)
    self.widgets[i+'pos'].connect("pressed", self['jog_'+i],1,True)
    self.widgets[i+'pos'].connect("released", self['jog_'+i],1,False)
self.widgets.jog_speed.connect("value_changed",self.jog_speed_changed)
```

Add these functions under the *HandlerClass* class:

```
def jog_x(self,widget,direction,state):
    self.gscreen.do_key_jog(_X,direction,state)
def jog_y(self,widget,direction,state):
    self.gscreen.do_key_jog(_Y,direction,state)
def jog_z(self,widget,direction,state):
    self.gscreen.do_key_jog(_Z,direction,state)
def jog_speed_changed(self,widget,value):
    self.gscreen.set_jog_rate(absolute = value)
```

Finally add two buttons to the GLADE file for each axis - one for positive, one for negative direction jogging. Name these buttons *xneg*, *xpos*, *yneg*, *ypos*, *zneg*, *zpos* respectively. add a *SpeedControl* widget to the GLADE file and name it *jog_speed*

5.5.5 Gscreen Start Up

Gscreen is really just infrastructure to load a custom GladeVCP file and interact with it.

1. Gscreen reads the options it was started with.
2. Gscreen sets the debug mode and set the optional skin name.
3. Gscreen checks to see if there are *local* XML, handler and/or locale files in the configuration folder. It will use them instead of the default ones (in *share/gscreen/skins/*) (There can be two separate screens displayed).
4. The main screen is loaded and translations set up. If present the second screen will be loaded and translations set up.

5. Optional Audio is initialized if available.
 6. It reads some of the INI file to initialize the units, and the number/type of axes.
 7. Initializes Python's binding to HAL to build a userspace component with the Gscreen name.
 8. GladeVCP's makepins is called to parse the XML file to build HAL pins for the HAL widgets and register the LinuxCNC connected widgets.
 9. Checks for a *local* handler file in the configuration folder or else uses the stock one from the skin folder.
 10. If there is a handler file gscreen parses it, and registers the function calls into Gscreen's namespace.
 11. Glade matches/registers all signal calls to functions in gscreen and the handler file.
 12. Gscreen checks the INI file for an option preference file name otherwise it uses *.gscreen_preferences* =.
 13. Gscreen checks to see if there is a preference function call (*initialize_preferences(self)*) in the handler file otherwise it uses the stock Gscreen one.
 14. Gscreen checks for ClassicLadder realtime component.
 15. Gscreen checks for the system wide GTK theme.
 16. Gscreen collects the jogging increments from the INI file.
 17. Gscreen collects the angular jogging increments from the INI file.
 18. Gscreen collects the default and max jog rate from the INI.
 19. Gscreen collects the max velocity of any axes from the INI's TRAJ section.
 20. Gscreen checks to see if there is angular axes then collects the default and max velocity from the INI file.
 21. Gscreen collect all the override setting from the INI.
 22. Gscreen checks if its a lathe configuration from the INI file.
 23. Gscreen finds the name of the tool_table, tool editor and param file from the INI.
 24. Gscreen checks the handler file for keybindings function (*initialize_keybindings(self)*) or else use Gscreen stock one.
 25. Gscreen checks the handler file for pins function (*initialize_pins(self)*) or else use Gscreen stock one.
 26. Gscreen checks the handler file for manual_toolchange function (*initialize_manual_toolchange(self)*) or else use Gscreen stock one.
 27. Gscreen checks the handler file for connect_signals function (*initialize_connect_signals(self)*) or else use Gscreen stock one.
 28. Gscreen checka the handler file for widgets function (*initialize_widgets(self)*) or else use Gscreen stock one.
 29. Gscreen seta up messages specified in the INI file.
 30. Gscreen tells HAL the Gscreen HAL component is finished making pins and is ready. If there is a terminal widget in the screen it will print all the Gscreen pins to it.
 31. Gscreen sets the display cycle time based on the INI file.
 32. Gscreen checks the handler file for *timer_interrupt(self)* function call otherwise use Gscreen's default function call.
-

5.5.6 INI Settings

Under the [DISPLAY] heading:

```
DISPLAY = gscreen -c tester
options:
  -d debugging on
  -v verbose debugging on
```

The -c switch allows one to select a *skin*. Gscreen assumes the Glade file and the handler file use this same name. The optional second screen will be the same name with a 2 (eg. tester2.glade) There is no second handler file allowed. It will only be loaded if it is present. Gscreen will search the LinuxCNC configuration file that was launched first for the files, then in the system skin folder.

5.5.7 User Dialog Messages

This function is used to display pop up dialog messages on the screen. These are defined in the INI file and controlled by HAL pins.

Boldtext is generally a title.

text is below that and usually longer.

Detail is hidden unless clicked on.

pinname is the basename of the HAL pins.

type specifies whether its a yes/no, ok, or status message.

Status messages will be shown in the status bar and the notify dialog. it requires no user intervention. ok messages require the user to click ok to close the dialog.

ok messages have one HAL pin to launch the dialog and one to signify it's waiting for response.

yes/no messages require the user to select yes or no buttons to close the dialog.

yes/no messages have three hal pins - one to show the dialog, one for waiting, and one for the answer.

Here is a sample INI code. It would be under the [DISPLAY] heading.

```
# This just shows in the status bar and desktop notify popup.
MESSAGE_BOLDTEXT = NONE
MESSAGE_TEXT = This is a statusbar test
MESSAGE_DETAILS = STATUS DETAILS
MESSAGE_TYPE = status
MESSAGE_PINNAME = statustest

# This will pop up a dialog that asks a yes no question
MESSAGE_BOLDTEXT = NONE
MESSAGE_TEXT = This is a yes no dialog test
MESSAGE_DETAILS = Y/N DETAILS
MESSAGE_TYPE = yesnodialog
MESSAGE_PINNAME = yndialogtest

# This pops up a dialog that requires an ok response and it shows in the status bar and
# the desktop notify popup.
MESSAGE_BOLDTEXT = This is the short text
MESSAGE_TEXT = This is the longer text of the both type test. It can be longer then the ↔
               status bar text
MESSAGE_DETAILS = BOTH DETAILS
MESSAGE_TYPE = okdialog status
MESSAGE_PINNAME = bothtest
```

5.5.7.1 Copy the Stock Handler/Glade File For Modification

If you wish to use a stock screen but modify it's handler file, you need to copy the stock file to your config file folder. Gscreen will see this and use the copied file. But where is the original file? If using a RIP linuxcnc the sample skins are in `/share/gscreen/skins/SCREENNAME` Installed versions of linuxcnc have them in slightly different places depending on the distribution used. An easy way to find the location is to open a terminal and start the sim screen you wish to use. In the terminal the file locations will be printed. It may help to add the `-d` switch to the gscreen load line in the INI.

Here is a sample:

```
chris@chris-ThinkPad-T500 ~/emc-dev/src $ linuxcnc
LINUXCNC - 2.7.14
Machine configuration directory is '/home/chris/emc-dev/configs/sim/gscreen/gscreen_custom'
Machine configuration file is 'industrial_lathe.ini'
Starting LinuxCNC...
Found file(lib): /home/chris/emc-dev/lib/hallib/core_sim.hal
Note: Using POSIX non-realtime
Found file(lib): /home/chris/emc-dev/lib/hallib/sim_spindle_encoder.hal
Found file(lib): /home/chris/emc-dev/lib/hallib/axis_manualtoolchange.hal
Found file(lib): /home/chris/emc-dev/lib/hallib/simulated_home.hal
**** GSCREEN WARNING: no audio alerts available - Is python-gst0.10 library installed?
**** GSCREEN INFO ini: /home/chris/emc-dev/configs/sim/gscreen/gscreen_custom/ ↵
    industrial_lathe.ini
**** GSCREEN INFO: Skin name = industrial

**** GSCREEN INFO: Using SKIN glade file from /home/chris/emc-dev/share/gscreen/skins/ ↵
    industrial/industrial.glade ****

**** GSCREEN INFO: No Screen 2 glade file present
**** GSCREEN INFO: handler file path: ['/home/chris/emc-dev/share/gscreen/skins/industrial/ ↵
    industrial_handler.py']
```

The line:

```
**** GSCREEN INFO: handler file path: ['/home/chris/emc-dev/share/gscreen/skins/industrial/ ↵
    industrial_handler.py']
```

shows where the stock file lives. Copy this file to your config folder.
This works the same for the Glade file.

5.6 TkLinuxCNC GUI

5.6.1 Introduction

TkLinuxCNC is one of the first graphical front-ends for LinuxCNC. It is written in Tcl and uses the Tk toolkit for the display. Being written in Tcl makes it very portable (it runs on a multitude of platforms). A separate backplot window can be displayed as shown.

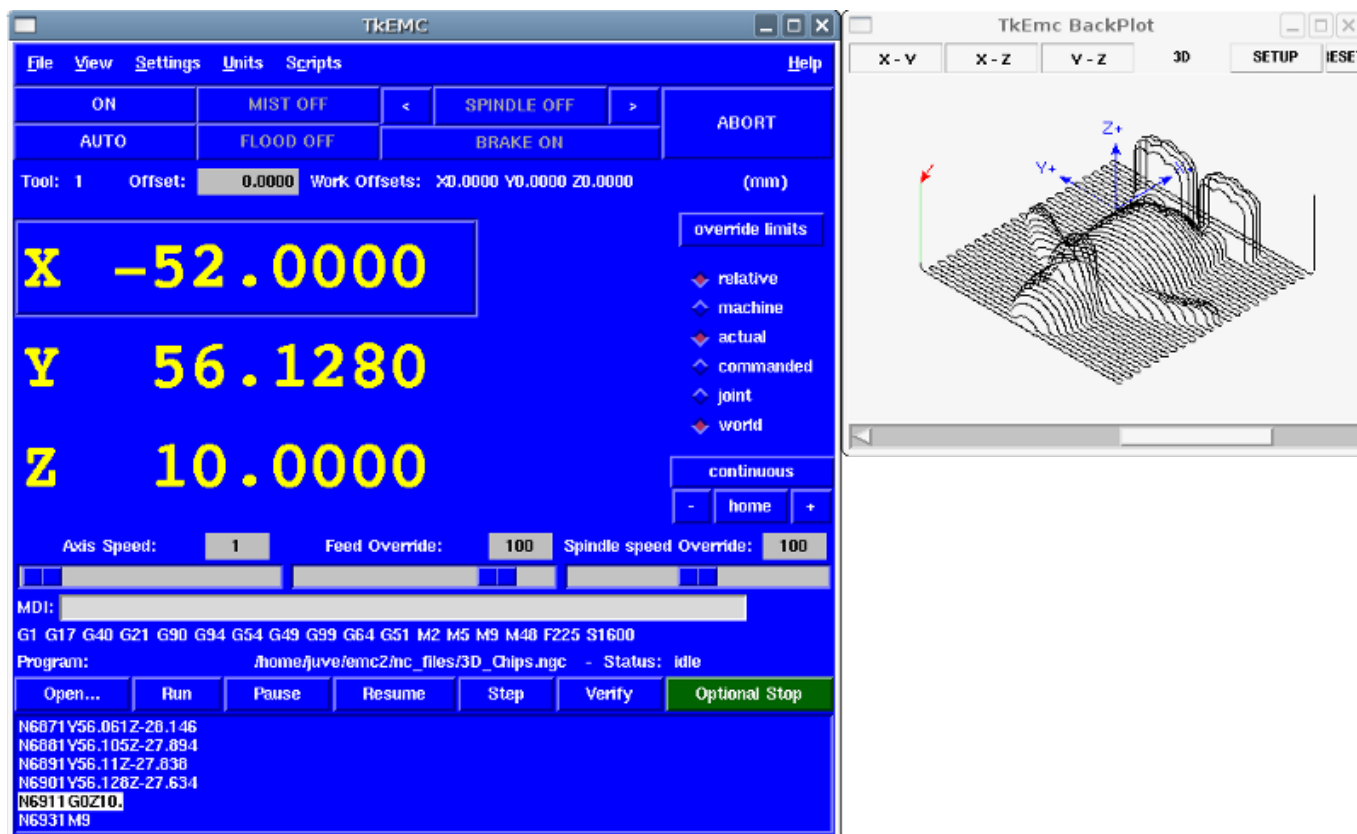


Figure 5.28: TkLinuxCNC Window

5.6.2 Getting Started

To select TkLinuxCNC as the front-end for LinuxCNC, edit the .ini file. In the section `[DISPLAY]` change the `DISPLAY` line to read

```
DISPLAY = tklinuxcnc
```

Then, start LinuxCNC and select that ini file. The sample configuration `sim/tklinuxcnc/tklinuxcnc.ini` is already configured to use TkLinuxCNC as its front-end.

When LinuxCNC is launched the [TKLinuxCNC window](#) is opened.

5.6.2.1 A typical session with TkLinuxCNC

1. Start LinuxCNC and select a configuration file.
2. Clear the *E-STOP* condition and turn the machine on (by pressing F1 then F2).
3. *Home* each axis.
4. Load the file to be milled.
5. Put the stock to be milled on the table.
6. Set the proper offsets for each axis by jogging and either homing again or right-clicking an axis name and entering an offset value. ¹

¹For some of these actions it might be necessary to change the mode LinuxCNC is currently running in.

7. Run the program.
8. To mill the same file again, return to step 6. To mill a different file, return to step 4. When you're done, exit LinuxCNC.

5.6.3 Elements of the TkLinuxCNC window

The TkLinuxCNC window contains the following elements:

- A menubar that allows you to perform various actions
- A set of buttons that allow you to change the current working mode, start/stop spindle and other relevant I/O
- Status bar for various offset related displays
- Coordinate display area
- A set of sliders which control *Jogging speed*, *Feed Override*, and *Spindle speed Override* which allow you to increase or decrease those settings
- Manual data input text box *MDI*
- Status bar display with active G-codes, M-codes, F- and S-words
- Interpreter related buttons
- A text display area that shows the G-code source of the loaded file

5.6.3.1 Main buttons

From left to right, the buttons are:

- Machine enable: *ESTOP* > *ESTOP RESET* > *ON*
- Toggle mist coolant
- Decrease spindle speed
- Set spindle direction *SPINDLE OFF* > *SPINDLE FORWARD* . *SPINDLE REVERSE*
- Increase spindle speed
- Abort

then on the second line:

- Operation mode: *MANUAL* > *MDI* > *AUTO*
- Toggle flood coolant
- Toggle spindle brake control

5.6.3.2 Offset display status bar

The Offset display status bar displays the currently selected tool (selected with Txx M6), the tool length offset (if active), and the work offsets (set by right-clicking the coordinates).

5.6.3.3 Coordinate Display Area

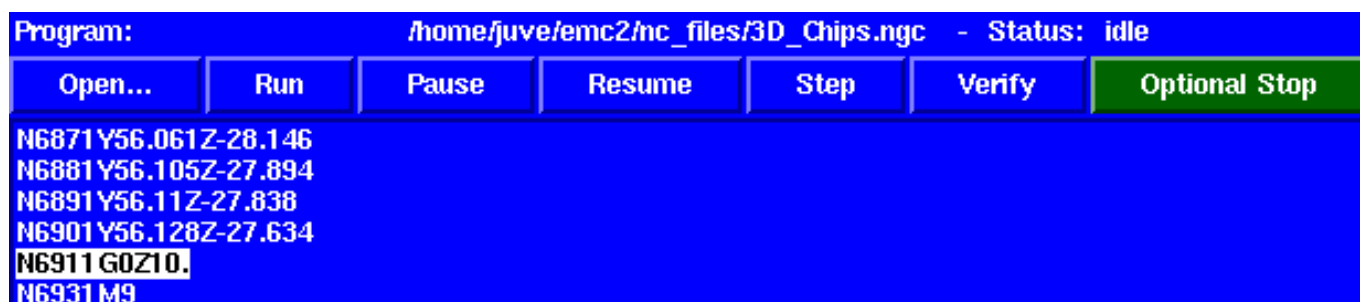
The main part of the display shows the current position of the tool. The color of the position readout depends on the state of the axis. If the axis is unhommed the axis will be displayed in yellow letters. Once homed it will be displayed in green letters. If there is an error with the current axis TkLinuxCNC will use red letter to show that. (for example if an hardware limit switch is tripped).

To properly interpret these numbers, refer to the radio boxes on the right. If the position is *Machine*, then the displayed number is in the machine coordinate system. If it is *Relative*, then the displayed number is in the offset coordinate system. Further down the choices can be *actual* or *commanded*. Actual refers to the feedback coming from encoders (if you have a servo machine), and the *commanded* refers to the position command send out to the motors. These values can differ for several reasons: Following error, deadband, encoder resolution, or step size. For instance, if you command a movement to X 0.0033 on your mill, but one step of your stepper motor is 0.00125, then the *Commanded* position will be 0.0033 but the *Actual* position will be 0.0025 (2 steps) or 0.00375 (3 steps).

Another set of radio buttons allows you to choose between *joint* and *world* view. These make little sense on a normal type of machine (e.g. trivial kinematics), but help on machines with non-trivial kinematics like robots or stewart platforms. (you can read more about kinematics in the Integrator Manual).

Backplot When the machine moves, it leaves a trail called the backplot. You can start the backplot window by selecting View→Backplot.

5.6.3.4 TkLinuxCNC Interpreter / Automatic Program Control



Control Buttons The buttons in the lower part of TkLinuxCNC are used to control the execution of a program:

+ * *Open* to load a program, * *Verify* to check it for errors, * *Run* to start the actual cutting, * *Pause* to stop it while running, * *Resume* to resume an already paused program, * *Step* to advance one line in the program and * *Optional Stop* to toggle the optional stop switch (if the button is green the program execution will be stopped on any M1 encountered).

Text Program Display Area When the program is running, the line currently being executed is highlighted in white. The text display will automatically scroll to show the current line.

5.6.3.5 Manual Control

Implicit keys TkLinuxCNC allows you to manually move the machine. This action is known as *jogging*. First, select the axis to be moved by clicking it. Then, click and hold the + or - button depending on the desired direction of motion. The first four axes can also be moved by the keyboard arrow keys (X and Y), the PAGE UP and PAGE DOWN keys (Z) and the [and] keys (A/4th).

+ If *Continuous* is selected, the motion will continue as long as the button or key is pressed. If another value is selected, the machine will move exactly the displayed distance each time the button is clicked or the key is pressed. The available values are:

+

1.0000, 0.1000, 0.0100, 0.0010, 0.0001

+ By pressing *Home* or the HOME key, the selected axis will be homed. Depending on your configuration, this may just set the axis value to be the absolute position 0.0, or it may make the machine move to a specific home location through use of *home switches*. See the [Homing Chapter](#) for more information.

+ By pressing *Override Limits*, the machine will temporarily be permitted to jog outside the limits defined in the .ini file. (Note: if *Override Limits* is active the button will be displayed using a red color).

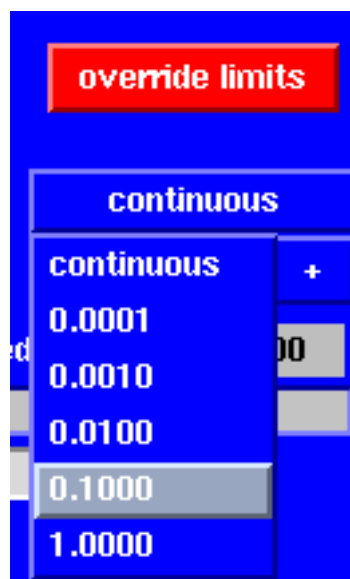


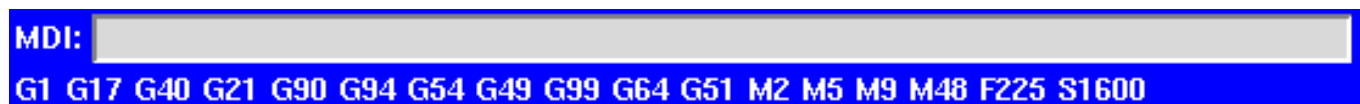
Figure 5.29: TkLinuxCNC Override Limits & Jogging increments example

The Spindle group The button on the first row selects the direction for the spindle to rotate: Counterclockwise, Stopped, Clockwise. The buttons next to it allow the user to increase or decrease the rotation speed. The button on the second row allows the spindle brake to be engaged or released. Depending on your machine configuration, not all the items in this group may have an effect.

The Coolant group The two buttons allow the *Mist* and *Flood* coolants to be turned on and off. Depending on your machine configuration, not all the items in this group may appear.

5.6.3.6 Code Entry

Manual Data Input (also called MDI), allows G-code programs to be entered manually, one line at a time. When the machine is not turned on, and not set to MDI mode, the code entry controls are unavailable.



This allows you to enter a G-code command to be executed. Execute the command by pressing Enter.

Active G-Codes This shows the *modal codes* that are active in the interpreter. For instance, *G54* indicates that the *G54 offset* is applied to all coordinates that are entered.

5.6.3.7 Jog Speed

By moving this slider, the speed of jogs can be modified. The numbers above refer to axis units / second. The text box with the number is clickable. Once clicked a popup window will appear, allowing for a number to be entered.

5.6.3.8 Feed Override

By moving this slider, the programmed feed rate can be modified. For instance, if a program requests *F60* and the slider is set to 120%, then the resulting feed rate will be 72. The text box with the number is clickable. Once clicked a popup window will appear, allowing for a number to be entered.

5.6.3.9 Spindle speed Override

The spindle speed override slider works exactly like the feed override slider, but it controls to the spindle speed. If a program requested *S500* (spindle speed 500 RPM), and the slider is set to 80%, then the resulting spindle speed will be 400 RPM. This slider has a minimum and maximum value defined in the ini file. If those are missing the slider is stuck at 100%. The text box with the number is clickable. Once clicked a popup window will appear, allowing for a number to be entered.

5.6.4 Keyboard Controls

Almost all actions in TkLinuxCNC can be accomplished with the keyboard. Many of the shortcuts are unavailable when in MDI mode.

The most frequently used keyboard shortcuts are shown in the following table.

Table 5.2: Most Common Keyboard Shortcuts

Keystroke	Action Taken
F1	Toggle Emergency Stop
F2	Turn machine on/off
, 1 .. 9, 0	Set feed override from 0% to 100%
X, ^	Activate first axis
Y, 1	Activate second axis
Z, 2	Activate third axis
A, 3	Activate fourth axis
Home	Send active axis Home
Left, Right	Jog first axis
Up, Down	Jog second axis
Pg Up, Pg Dn	Jog third axis
[,]	Jog fourth axis
ESC	Stop execution

5.7 QtPlasmaC

5.7.1 Preamble

Except where noted, this guide assumes the user is using the latest version of QtPlasmaC. Version history can be seen by visiting this [link](#) which will show the latest available version.

The installed QtPlasmaC version is displayed in the title bar. See [Update QtPlasmaC](#) for information on updating QtPlasmaC.

5.7.2 License

QtPlasmaC and all of its related software are released under GPLv2.

5.7.3 Introduction

QtPlasmaC is a GUI for plasma cutting which utilises the [plasmac component](#) for controlling a plasma table from LinuxCNC v2.9 or later using the Debian Buster or similar distribution.

The QtPlasmaC GUI supports up to five axes and uses the QtVCP infrastructure provided with LinuxCNC.

The standard theme is based on a design by user "pinder" on the LinuxCNC Forum and the colors are able to be changed by the user.

The QtPlasmaC GUI will run on any hardware that is supported by LinuxCNC provided there are enough hardware I/O pins to fulfill the requirements of a plasma configuration.

There are three available formats:

- 16:9 with a minimum resolution of 1366 x 768
- 9:16 with a minimum resolution of 768 x 1366
- 4:3 with a minimum resolution of 1024 x 768

Screenshot examples of QtPlasmaC are below:

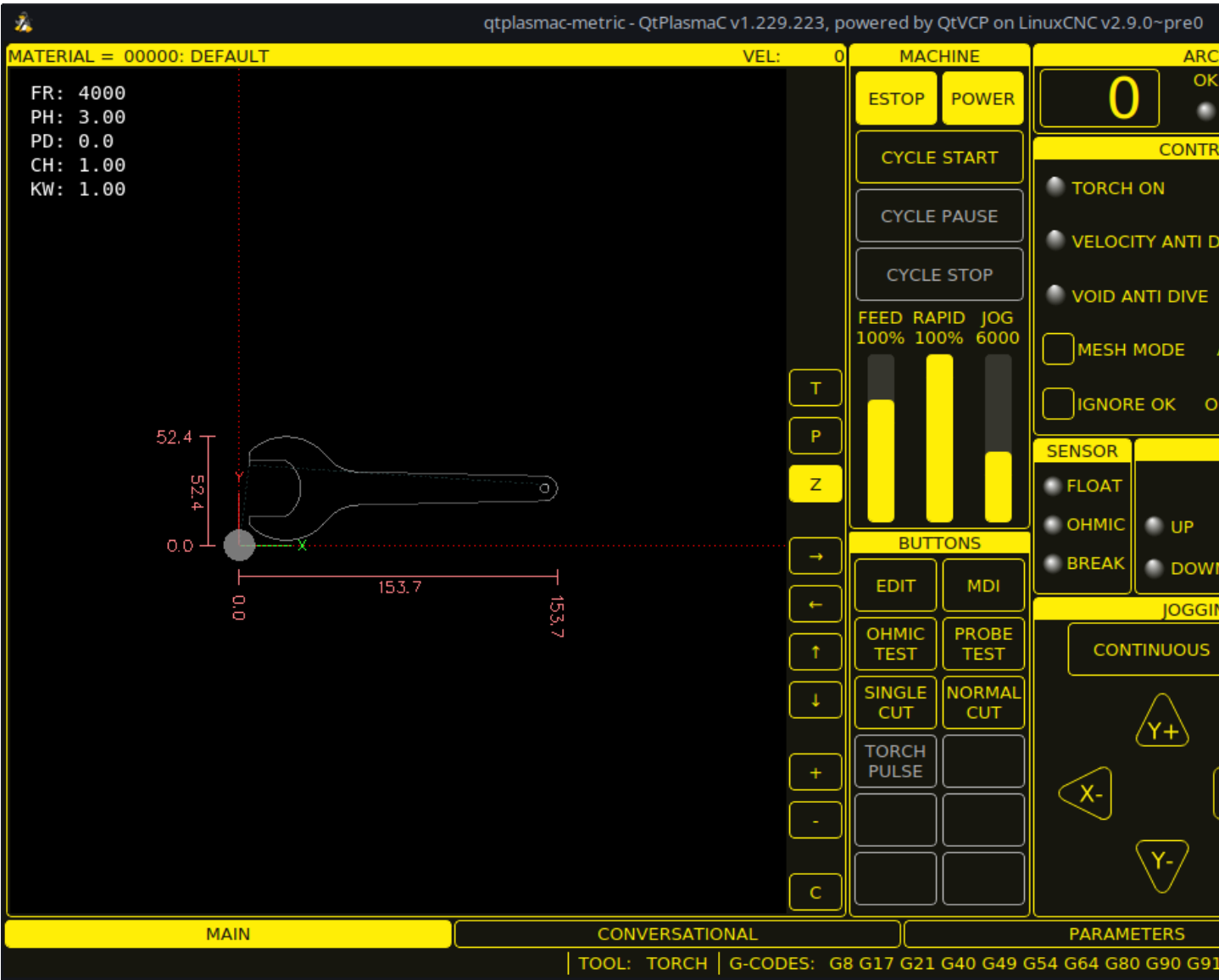


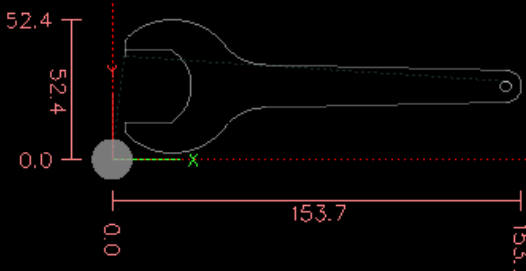
Figure 5.30: 16:9

qtplasmac-imperial - QtPlasmaC v1.229.222, powered by QtVCP on LinuxCNC v2.9.0~pre0

BUTTONS

MATERIAL = 00000: DEFAULT VEL: 0

FR: 3988
PH: 3.05
PD: 0.0
CH: 1.02
KW: 1.02



52.4
52.4
0.0
153.7
0.0

SINGLE CUT
TORCH PULSE
NORMAL CUT
PROBE TEST
OHMIC TEST

C + - ↑ ↓ ← → Z P T

FILE EDIT MDI

MACHINE ESTOP POWER CYCLE START CYCLE PAUSE CYCLE STOP

FEED RAPID JOG
100% 100% 240

DRO HOME ALL WCS G54 CAMERA LASER X0Y0

HOME X 0.000 0
HOME Y 0.000 0
HOME Z 71.120 0

JOGGING CONTINUOUS FAST

Y+ Z+ X- X+ Y- Z-

THC ENABLE ENABLED ACTIVE UP DOWN

SENSOR FLOAT OHMIC BREAK

ARC CLEAR metric_wrench.ngc RELOAD

```
1 ;metric wrench
2
3 #<holes>=4
4
5 g21
6 g64p0.05
7 m52p1
8
9 f[#<_hal[plasmac.cut-feed-r
10
```

0 OK OVERRIDE - 0.00 +

CONTROL

TORCH ON ENABLE
VELOCITY ANTI DIVE ENABLE
VOID ANTI DIVE ENABLE
MESH MODE AUTO VOLTS
IGNORE OK OHMIC ENABLE

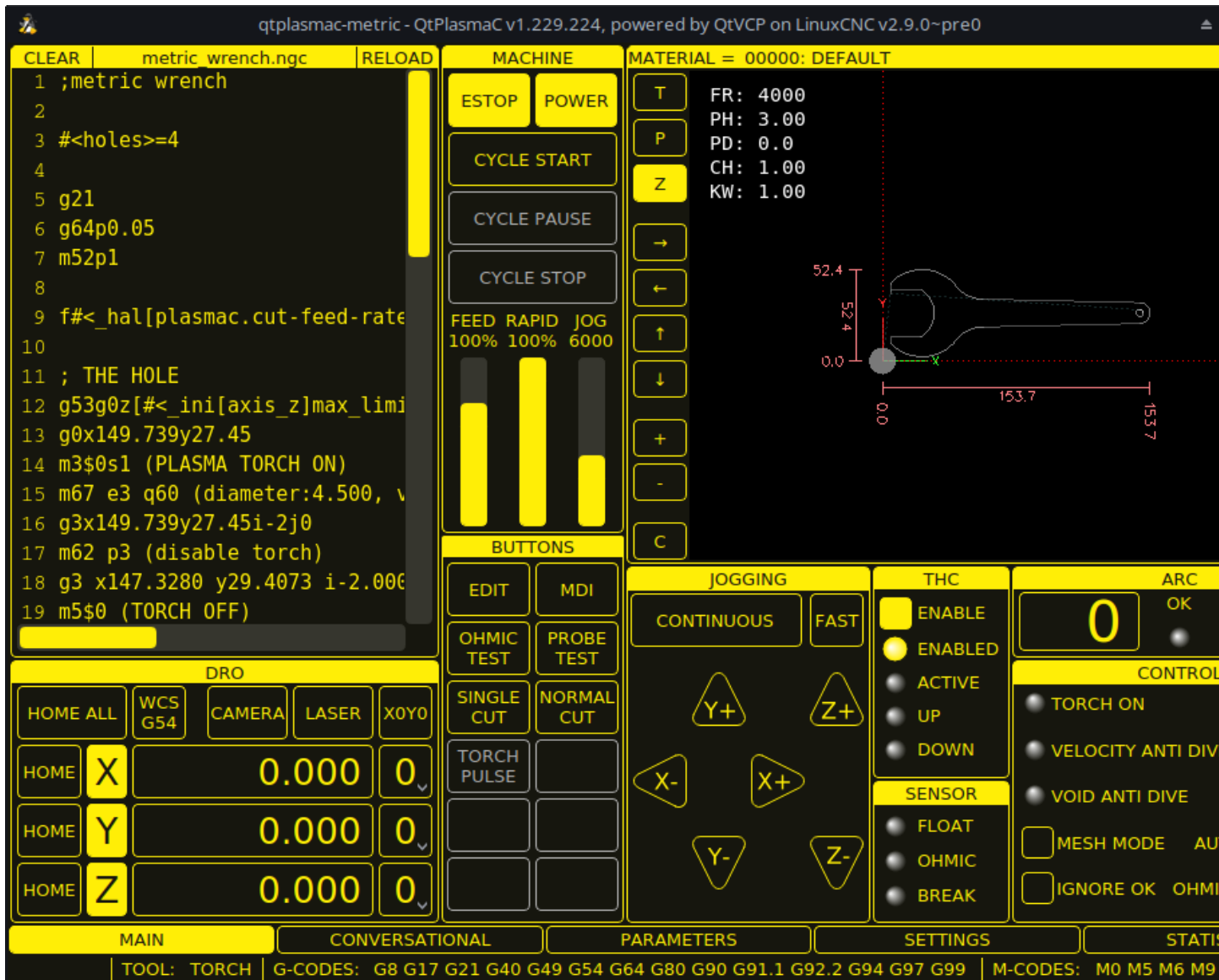


Figure 5.32: 4:3

5.7.4 Installing LinuxCNC

The preferred method for installing LinuxCNC is via an ISO image as described below.

Note

It is possible to install and run LinuxCNC on a variety of Linux distributions however that is beyond the scope of this User Guide. If the user wishes to install a Linux distribution other than those recommended, they will first need to install their preferred Linux distribution and then install LinuxCNC v2.9 or later along with any required dependencies.

5.7.4.1 If The User Does Not Have Linux Installed

Installation instructions are available at: <link:../getting-started/getting-linuxcnc.html>

Following these instructions will yield a machine with the current stable branch (v2.8) of LinuxCNC on Debian Buster.

5.7.4.2 Package Installation (Buildbot) If The User Has Linux with LinuxCNC v2.8

A package installation (Buildbot) uses prebuilt packages from the LinuxCNC Buildbot, instructions for upgrading from 2.8 to 2.9 are available at: <http://buildbot.linuxcnc.org>

Following these instructions by using the below stanzas will upgrade the machine to the last LinuxCNC Buildbot build master branch (v2.9) of LinuxCNC. This may not always be the latest version of master branch (v2.9) as from time to time the LinuxCNC Buildbot may stop due to errors.

```
deb      http://buildbot.linuxcnc.org/ buster master-rtpreempt
deb-src  http://buildbot.linuxcnc.org/ buster master-rtpreempt
```

5.7.4.3 Run In Place Installation If The User Has Linux with LinuxCNC v2.8

A run in place installation runs LinuxCNC from a locally compiled version usually located at `~/linuxcnc-dev`, instructions for building a run in place installation are available at: link:../code/building-linuxcnc.html

Following these instructions will install the latest master branch (v2.9) of LinuxCNC.

5.7.5 Creating A QtPlasmaC Configuration

Prior to creating a QtPlasmaC configuration, it is important that the user has a firm understanding of the operating modes available, as well as the I/O's that are required for successful plasma operation.

5.7.5.1 Modes

QtPlasmaC requires the selection of one of following three operating modes:

Mode	Description
0	Uses an external arc voltage input to calculate both Arc Voltage (for Torch Height Control) and Arc OK.
1	Uses an external arc voltage input to calculate Arc Voltage (for Torch Height Control). Uses an external Arc OK input for Arc OK.
2	Uses an external Arc OK input for Arc OK. Use external up/down signals for Torch Height Control.



Important

If the plasma power source has an Arc OK (Transfer) output then it is recommended to use that for Arc OK rather than the soft (calculated) Arc OK provided by mode 0. It may also be possible to use a [reed relay](#) as an alternative method to establish an Arc OK signal when the power source does not provide one.

Note

For fine tuning of Mode 0 Arc OK see [Tuning Mode 0 Arc OK](#) in the Advanced Topics section of the manual.

5.7.5.2 Available I/Os

Note

This section only touches on the hardware I/O's required for QtPlasmaC. Base machine requirements such as limit switches, home switches, etc. are in addition to these.

Name	Modes	Description
Arc Voltage	0, 1	Analog input; optional . HAL pin name plasmac.arc-voltage-in Connected to the velocity output of an encoder equipped breakout board. This signal is used to read the arc voltage to determine the necessary corrections to maintain the torch distance from the work piece during cutting.
Arc OK	1, 2	Digital input; optional . HAL pin name plasmac.arc-ok-in Connected from the Arc OK output of the plasma power source to an input on the breakout board. This signal is used to determine if the cutting arc has been established and it is ok for the machine to move (sometimes called arc transfer).
Float Switch	0, 1, 2	Digital input; optional, see info below table : HAL pin name plasmac.float-switch Connected from a breakout board input to a switch on the floating head. This signal is used to mechanically probe the work piece with the torch and set Z zero at the top of the work piece. If used and no ohmic probe is configured, this is the probing method. If used and an ohmic probe is configured, this is the fallback probing method.
Ohmic Probe	0, 1, 2	Digital input; optional, see info below table : HAL pin name plasmac.ohmic-probe Connected from to the ohmic probe's output to a breakout board input. This signal is used to probe electronically by completing a circuit using the work piece and the torch consumables and set Z zero at the top of the work piece. If used, this is the primary probing method. If an ohmic probe fails to locate the work piece, and there is no float switch is present, probing will continue until the torch breaks away or the minimum Z limit is reached.
Ohmic Probe Enable	0, 1, 2	Digital output; optional, see info below table : HAL pin name plasmac.ohmic-enable Connected from a breakout board output to an input to control the ohmic probe's power.
Breakaway Switch	0, 1, 2	Digital input; optional, see info below table : HAL pin name plasmac.breakaway Connected from a breakout board input to a torch breakaway detection switch. This signal senses if the torch has broken away from its cradle.
Torch On	0, 1, 2	Digital output; required . HAL pin name plasmac.torch-on Connected from a breakout board output to the torch-on input of the plasma power supply. This signal is used to control the plasma power supply and start the arc.

Move Up	2	Digital input; optional . HAL pin name plasmac.move-up Connected from the up output of the external THC control to a break out board input. This signal is used to control the Z axis in an upward motion and make necessary corrections to maintain the torch distance from the work piece during cutting.
Move Down	2	Digital input; optional . HAL pin name plasmac.move-down Connected from the down output of the external THC control to a break out board input. This signal is used to control the Z axis in a downward motion and make necessary corrections to maintain the torch distance from the work piece during cutting.
Scribe Arming	0, 1, 2	Digital output; optional . HAL pin name plasmac.scribe-arm Connected from a breakout board output to the scribe arming circuit. This signal is used to place the scribe into position on the work piece .
Scribe On	0, 1, 2	Digital output; optional . HAL pin name plasmac.scribe-on Connected from a breakout board output to the scribe-on circuit. This signal is used to turn the scribing device on.
Laser On	0, 1, 2	Digital output; optional . HAL pin name qtplasmac.laser_on This signal is used to turn the alignment laser on.

Only one of either **Float Switch** or **Ohmic Probe** is required. If both are used then **Float Switch** will be a fallback if **Ohmic Probe** is not sensed.

If **Ohmic Probe** is used then **Ohmic Probe Enable** is required to be checked on the QtPlasmaC GUI.

Breakaway Switch is not mandatory because the **Float Switch** is treated the same as a breakaway when not probing. If they are two separate switches, and there are not enough inputs on the breakout board, they could be combined and connected as a **Float Switch**.

Note

The minimum I/O requirement for a QtPlasmaC configuration to function are: **Arc Voltage** input OR **Arc OK** input, **Float Switch** input, and **Torch On** output. To reiterate, in this case QtPlasmaC will treat the float switch as a breakaway switch when it is not probing.

5.7.5.3 Recommended Settings:

Refer to the [Heights Diagram](#) diagram for a visual representation of the terms below.

- **[AXIS_Z] MIN_LIMIT** should be just below top of the slats with allowances for float_switch_travel and over travel tolerance. For example, if the user's float switch takes 4mm (0.157") to activate then set the Z minimum to 5mm (0.2") plus an allowance for overrun (either calculated using the equation below or allow 5mm (0.2") below the lowest slat).
 - **[AXIS_Z] MAX_LIMIT** should be the highest the user wants the Z axis to travel (it must not be lower than Z_HOME_OFFSET).
 - **[AXIS_Z] HOME** should be set to be approximately 5mm-10mm (0.2"-0.4") below the maximum limit.
 - **Floating Head** - it is recommended that a floating head be used and that it has enough movement to allow for overrun during probing. Overrun can be calculated using the following formula:
-

```
o = 0.5 * a * (v / a)^2
```

where: o = overrun, a = acceleration in units/sec² and v = velocity in units/sec.

Metric example: given a Z axis MAX_ACCELERATION of 600mm/s² and MAX_VELOCITY of 60mm/s, the overrun would be 3mm.

Imperial example: given a Z axis MAX_ACCELERATION of 24in/s² and MAX_VELOCITY of 2.4in/s, the overrun would be 0.12in.

On machines that will utilize an ohmic probe as the primary method of probing, it is highly recommended to install a switch on the floating head as a backup means of stopping Z motion in the event of ohmic probe failure due to dirty surfaces.

5.7.5.4 Configuring

LinuxCNC provides two configuration wizards which can be used to build a machine configuration. The choice of these wizards is dependent on the hardware used to control the machine.

If the user wishes to use a Run In Place installation then prior to running one of the following commands they will need to run the following command from a terminal:

```
source ~/linuxcnc-dev/scripts/rip-environment
```

If using a Package installation then no additional action is required.

If using a parallel port, use the [StepConf wizard](#) (enter the following command into a terminal window):

```
stepconf
```

If using a Mesa Electronics board, use the [PnCConf wizard](#) (enter the following command into a terminal window):

```
pncconf
```

If using a Pico Systems board:

[This LinuxCNC forum thread](#) may be helpful.

The machine specific settings are not described here, refer to the documentation for the particular configuration wizard that is being used.

There are Linuxcnc forum sections available for these wizards:

[StepConf Wizard](#)

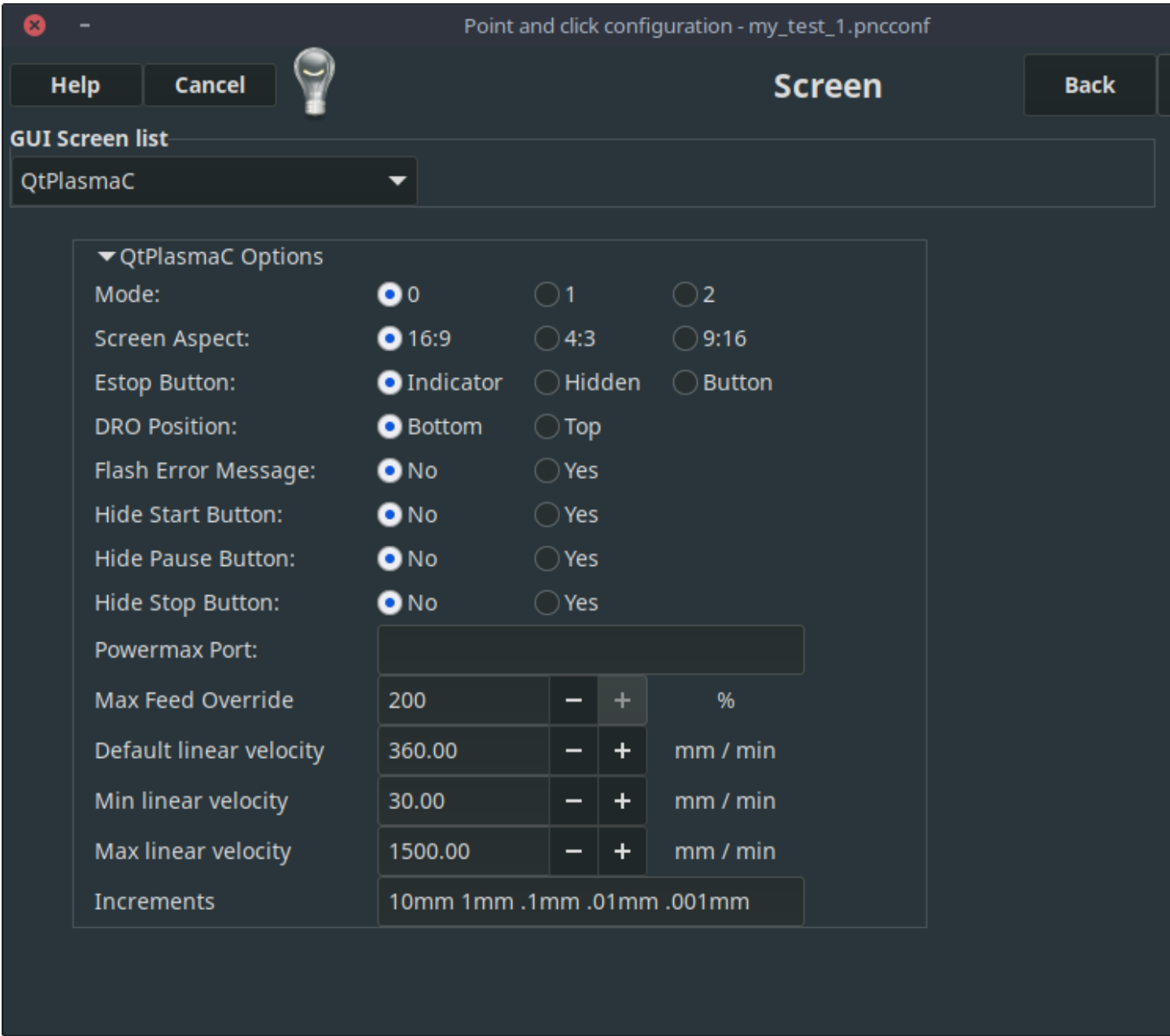
[PnCConf Wizard](#)

Fill in the required entries to suit the machine wiring/breakout board configuration.

QtPlasmaC adds two pages to the LinuxCNC configuration wizards for QtPlasmaC specific parameters, the two pages are QtPlasmaC options and [User Buttons](#). Complete each of the wizards QtPlasmaC page to suit the machine that is being configured and the user button requirements.

Note that PnCConf options allow user selection of Feed Override, Linear Velocity, and Jog Increments whereas in StepConf these are automatically calculated and set.

PnCConf QtPlasmaC Options:



StepConf QtPlasmaC Options:

Stepconf -Stepper Configuration Wizard

Cancel  Options Back Forward

Screen: QtPlasmaC ▼

Mode: ☒ 0 ☐ 1 ☐ 2

Screen Aspect: ☒ 16:9 ☐ 4:3 ☐ 9:16

Estop Button: ☒ Indicator ☐ Hidden ☐ Button

DRO Position: ☒ Bottom ☐ Top

Flash Error Message: ☒ No ☐ Yes

Hide Start Button: ☒ No ☐ Yes

Hide Pause Button: ☒ No ☐ Yes


Hide Stop Button: ☒ No ☐ Yes

Powermax Port:

☐ Include Classicladder PLC
 ▶ Set Ladder Options

QtPlasmaC User Buttons:

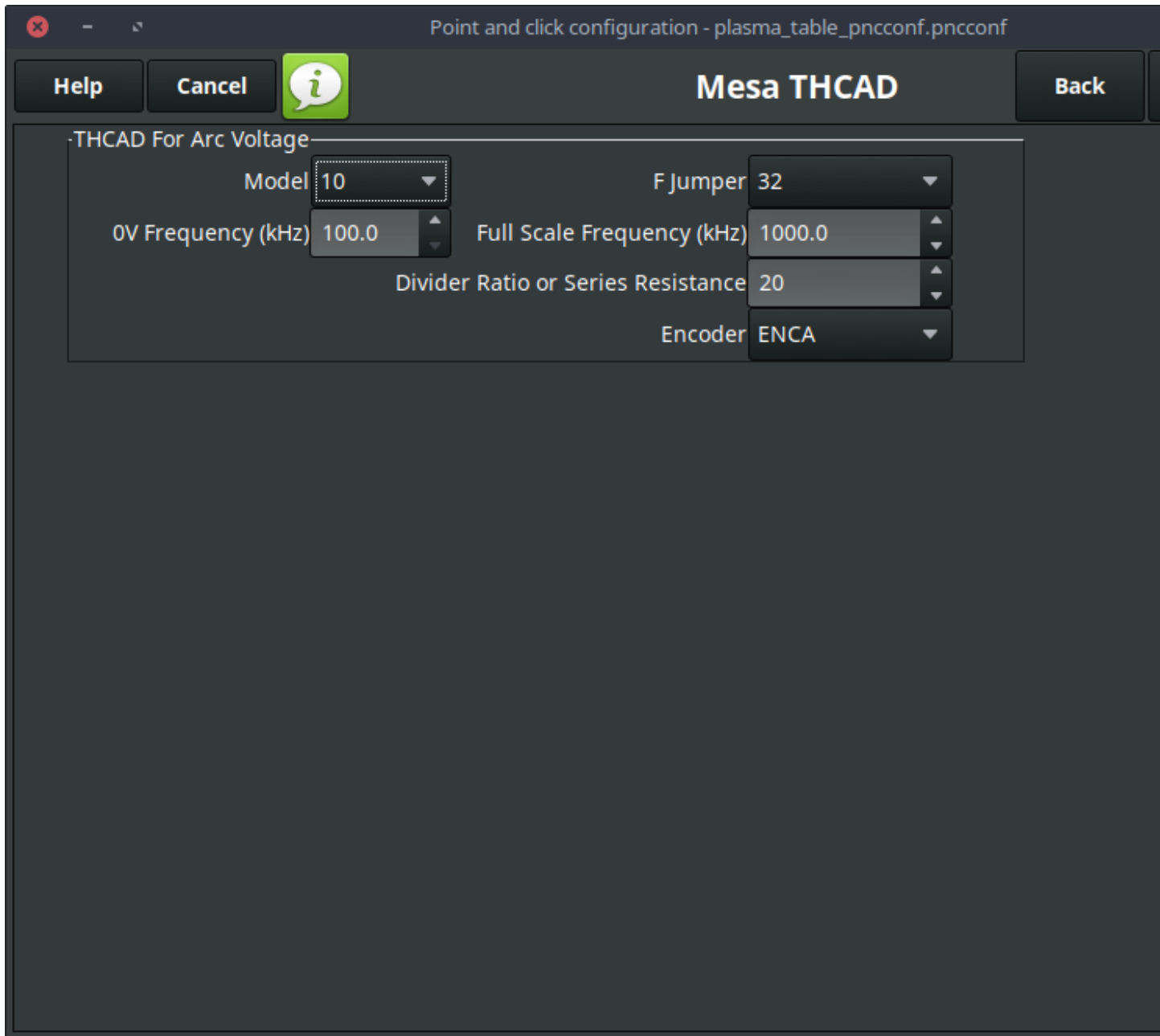
Point and click configuration - my_LinuxCNC_machine0.pncconf

Help Cancel  **QtPlasmaC User Buttons** Back

NUM	NAME	CODE
1	PROBE\TEST	probe-test 10
2	OHMIC\TEST	ohmic-test
3	SINGLE\CUT	single-cut
4	NORMAL\CUT	cut-type
5	TORCH\PULSE	torch-pulse 0.5
6		
7		
8		
9		
10		
11		
12		
13		
14		
15		
16		
17		
18		
19		
20		

QtPlasmaC THCAD:

The THCAD screen will only appear if a Plasma Encoder is selected in the card screen.



More information on [Mesa THCAD](#).

When the configuration is complete, the wizard will save a copy of the configuration that may be loaded and edited at a later time, a working QtPlasmaC configuration will be created in the following directory: `~/linuxcnc/configs/<machine_name>`.

The newly created QtPlasmaC configuration can be run by entering the following command into a terminal window (**change "`<machine_name>`" to the machine name entered into the configuration wizard**):

For a package installation (Buildbot):

```
linuxcnc ~/linuxcnc/configs/<machine_name>/<machine_name>.ini
```

For a run in place installation:

```
~/linuxcnc-dev/scripts/linuxcnc ~/linuxcnc/configs/<machine_name>/<machine_name>.ini
```


After running the above command LinuxCNC should be running with the QtPlasmaC GUI visible.

**Important**

BEFORE PROCEEDING, THE USER SHOULD BE ABLE TO HOME THE MACHINE, ZERO EACH AXIS, JOG ALL AXES TO SOFT LIMITS WITHOUT CRASHING, AND RUN TEST G-CODE PROGRAMS WITHOUT ANY ERRORS.

ONLY WHEN this criteria is met should the user proceed with the QtPlasmaC initial setup.

Note

It is possible to create a sim configuration using StepConf but it is not possible to have tandem joints in the sim configuration.

5.7.5.5 Qt Dependency Errors

If any Qt dependency errors are encountered while attempting to run the QtPlasmaC configuration, the user may need to run the QtVCP installation script to resolve these issues.

For a package installation (Buildbot) enter the following command in a terminal window:

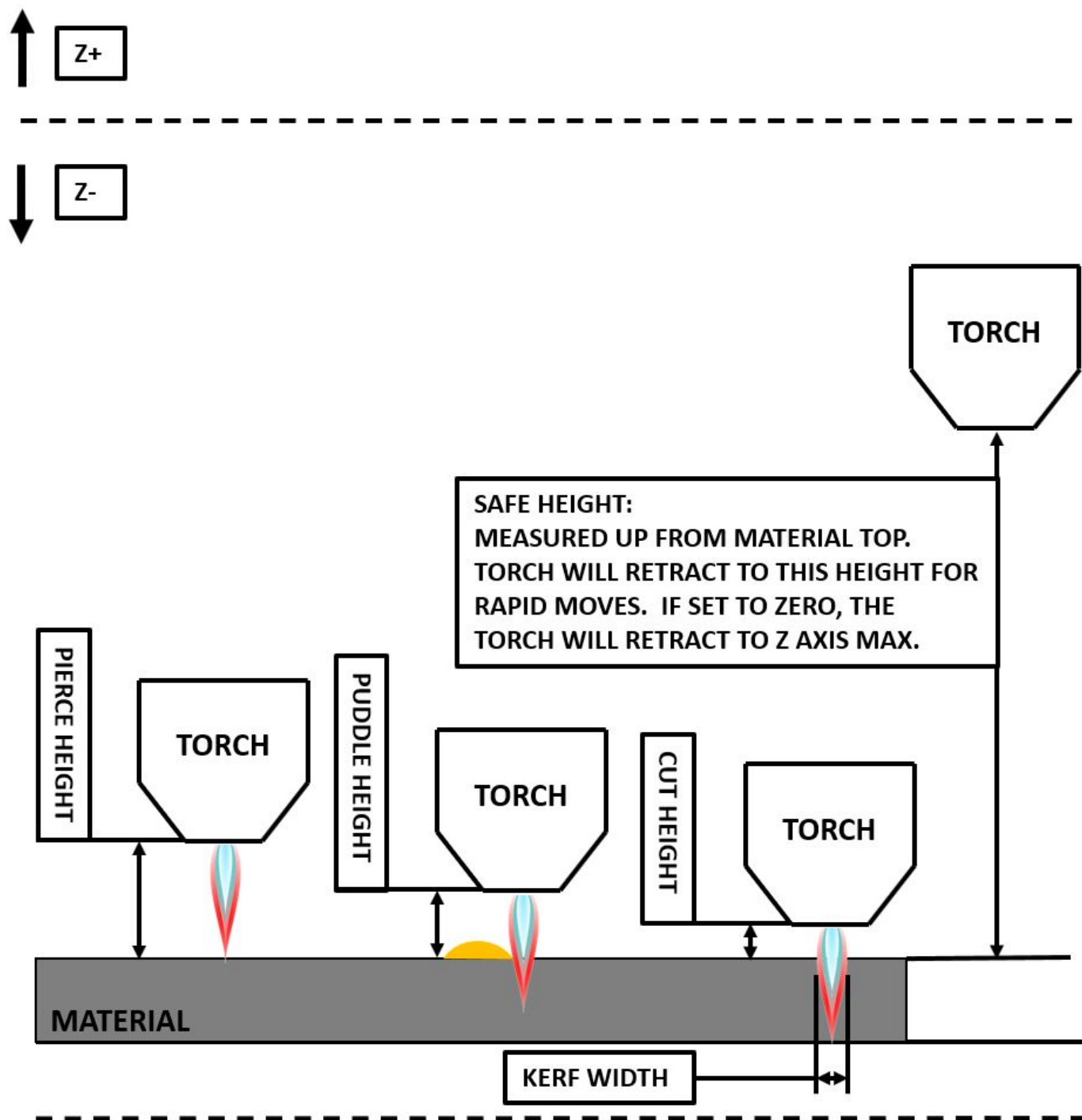
```
/usr/lib/python3/dist-packages/qtvc/designer/install_script
```

For a run in place installation enter the following command in a terminal window:

```
~/linuxcnc-dev/lib/python/qtvc/designer/install_script
```

5.7.5.6 Initial Setup

The following heights diagram will help the user visualize the different heights involved in plasma cutting and how they are measured:



Click on the [Parameters Tab](#) to view the **CONFIGURATION** section which shows the user settable parameters. It is necessary to ensure every one of these settings is tailored to the machine.

To set the Z axis DRO relative to the Z axis `MINIMUM_LIMIT`, the user should perform the following steps. It is important to understand that in QtPlasmaC, touching off the Z axis DRO has no effect on the Z axis position while running a G-code program. These steps simply allow the user to more easily

set the probe height as after performing the steps, the displayed Z axis DRO value will be relative to Z axis `MINIMUM_LIMIT`.

1. The user should be familiar with the recommended [Z Axis Settings](#).
2. Home the Z axis.
3. Ensure there is nothing below the torch then jog the Z axis down until it stops at the Z axis `MINIMUM_LIMIT` then click the 0 next to the Z axis DRO to **Touch Off** with the Z axis selected to set the Z axis at zero offset. This step only serves to allow the user to more easily visualize and adjust **Probe Height** this value is measured from the Z axis `MINIMUM_LIMIT` up.
4. Home the Z axis again.

If the machine is equipped with a float switch then the user will need to set the offset in the **CONFIGURATION** section of the **PARAMETERS** tab. This will be done by running a "Probe Test" cycle.

1. Check that the Probe Speed and the Probe Height in the **CONFIGURATION** section of the **PARAMETERS** tab are correct. QtPlasmaC can probe at the full Z axis velocity so long as the machine has enough movement in the float switch to absorb any overrun. If the machine is suitable, the user could set the Probe Height to a value near the Z axis minimum and do all probing at full speed.
2. If the machine is not already homed and in the home position, home the machine.
3. Place some material on the slats under the torch.
4. Press the **PROBE TEST** button.
5. The Z axis will probe down, find the material then move up to the specified **Pierce Height** as set by the currently selected material. The torch will wait in this position for the time set in the `<machine_name>.prefs` file. The default probe test hold time is 10 seconds, this value may be edited in the `<machine_name>.prefs` file. After this the torch will return to the starting height.
6. Measure the distance between the material and the tip of the torch while the torch is waiting at **Pierce Height**.
7. If the measurement is greater than the **Pierce Height** of the currently selected material, then reduce the "Float Travel" in the **CONFIGURATION** section of the **PARAMETERS** tab by the difference between the measured value and the specified value. If the measurement is less than **Pierce Height** of the currently selected material, then increase the "Float Travel" in the **CONFIGURATION** section of the **PARAMETERS** tab by the difference between the specified value and the measured value.
8. After the adjustments to the "Float Travel" have been made, repeat the process from #4 above until the measured distance between the material and the torch tip matches the **Pierce Height** of the currently selected material.
9. If the table has a laser or camera for sheet alignment, a scribe, or uses offset probing then the required offsets need to be applied by following the procedure described in [Peripheral Offsets](#).
10. CONGRATULATIONS! The user should now have a working QtPlasmaC Configuration.

Note

If the amount of time between the torch contacting the material and when the torch moves up and comes to rest at the Pierce Height seems excessive, see [the probing section](#) for a possible solution.

**Important**

IF USING A **Mesa Electronics THCAD** THEN THE **Voltage Scale** VALUE WAS OBTAINED MATHEMATICALLY. IF THE USER INTENDS TO USE CUT VOLTAGES FROM A MANUFACTURE'S CUT CHART THEN IT WOULD BE ADVISABLE TO DO MEASUREMENTS OF ACTUAL VOLTAGES AND FINE TUNE THE **Voltage Scale** AND **Voltage Offset**.

**Caution**

PLASMA CUTTING VOLTAGES CAN BE LETHAL, IF THE USER IS NOT EXPERIENCED IN DOING THESE MEASUREMENTS GET SOME QUALIFIED HELP.

5.7.6 Migrating to QtPlasmaC From PlasmaC (Axis or Gmoccapy)

There are two methods available to get from a working PlasmaC configuration to a new QtPlasmaC configuration. These methods assume the user is on LinuxCNC v2.9 or later, QtVCP is installed, and all dependency requirements are satisfied.

If there are Qt dependency errors, the user should run the [QtVCP install script](#).

5.7.6.1 Quick Method

A quick method to move to QtPlasmaC from PlasmaC (loaded on top of either Axis or Gmoccapy) is to use the `plasmac2qt` conversion program which will attempt to create a new QtPlasmaC configuration from an existing PlasmaC .ini file. This program will convert the user's parameters, settings, and materials from the previous PlasmaC configuration and create a new QtPlasmaC configuration directory in the `~/linuxcnc/configs` directory.

This methods will keep the original PlasmaC config as a backup with `_plasmac` and a time stamp appended to the directory name.

To run the `plasmac2qt` conversion program, use the following instructions:

For a package installation (Buildbot) enter the following line in a terminal window:

```
qtplasmac-plasmac2qt
```

For a run in place installation enter the following lines in terminal window:

```
source ~/linuxcnc-dev/scripts/rip-environment
qtplasmac-plasmac2qt
```

The following screen will be displayed:

PlasmaC2Qt

Convert Existing PlasmaC Configuration To A New QtPlasmaC Configuration

INI FILE IN EXISTING PLASMAC CONFIG:

☒ SELECT

MONITOR ASPECT RATIO:

☒ 16:9 ☐ 9:16 ☐ 4:3

ESTOP IS AN INDICATOR ONLY

☒ ESTOP: 0 ☐ ESTOP: 1 ☐ ESTOP: 2

OPTIONAL:

Laser On HAL pin: (bit output)

Mandatory Settings

Field	Description	Examples
INI FILE IN EXISTING PLASMAC CONFIG	This is the .ini file of the PlasmaC config that requires migrating.	<machine_name>.ini
MONITOR ASPECT RATIO	This is the aspect ratio format for the GUI.	16:9
ESTOP	Selects the required Estop type based on the following criteria: 0 - Estop is an indicator only. 1 - Estop indicator is hidden. 2 - Estop is a button.	ESTOP:1

Optional Setting

This setting is not required unless the machine has a [laser](#) for sheet alignment.

Leave this blank if it is not used/required.

Field	Description	Examples
Laser On HAL Pin	Power on a laser crosshair for sheet alignment.	Parallel Port Example: parport.0.pin-16-out Mesa 7i96 Example: hm2_7i96.0.ssr.00.out-00

After filling in the appropriate entries, press **CONVERT**.

Note

This method will not change any existing debounce components to the new dbounce component. If the user wishes to change to the new dbounce component then the New Base Config method should be used for migration.

5.7.6.2 New Base Config Method

This method to move to QtPlasmaC from PlasmaC (loaded on top of either Axis or Gmoccapy) is to use a [configuration wizard](#) to create a new configuration. This method then allows changing of the base machine configuration at a later date via the configuration wizard provided that the base ini and base HAL files have not been edited.

This method requires that the user take note of all HAL pins used in the existing config so they can be entered into the configuration wizard. Any custom HAL commands will also need to be noted and added manually to either the custom.hal file or the custom_postgui.hal file which will be created by the configuration wizard.

After using the wizard, the user can then run a conversion program (cfg2prefs) to convert the parameters, settings, and materials from the previous PlasmaC configuration to the new QtPlasmaC configuration. This tool should be used immediately after the user has created a new QtPlasmaC configuration.

Prior to running this conversion program, it is mandatory that the user have both an existing PlasmaC configuration and a new QtPlasmaC configuration. This program WILL overwrite the existing QtPlasmaC preferences and materials files, and should be used with caution if it is not being run on a new QtPlasmaC configuration.

The program will create a time stamped backup of the original preferences file and the existing materials file (if it exists).

It will read the existing <machine_name>_config.cfg, <machine_name>_run.cfg, <machine_name>_wizard and plasmac_stats.var files and write them to an existing <machine_name>_prefs file. It will also copy the <machine_name>_material.cfg file to the existing QtPlasmaC configuration.

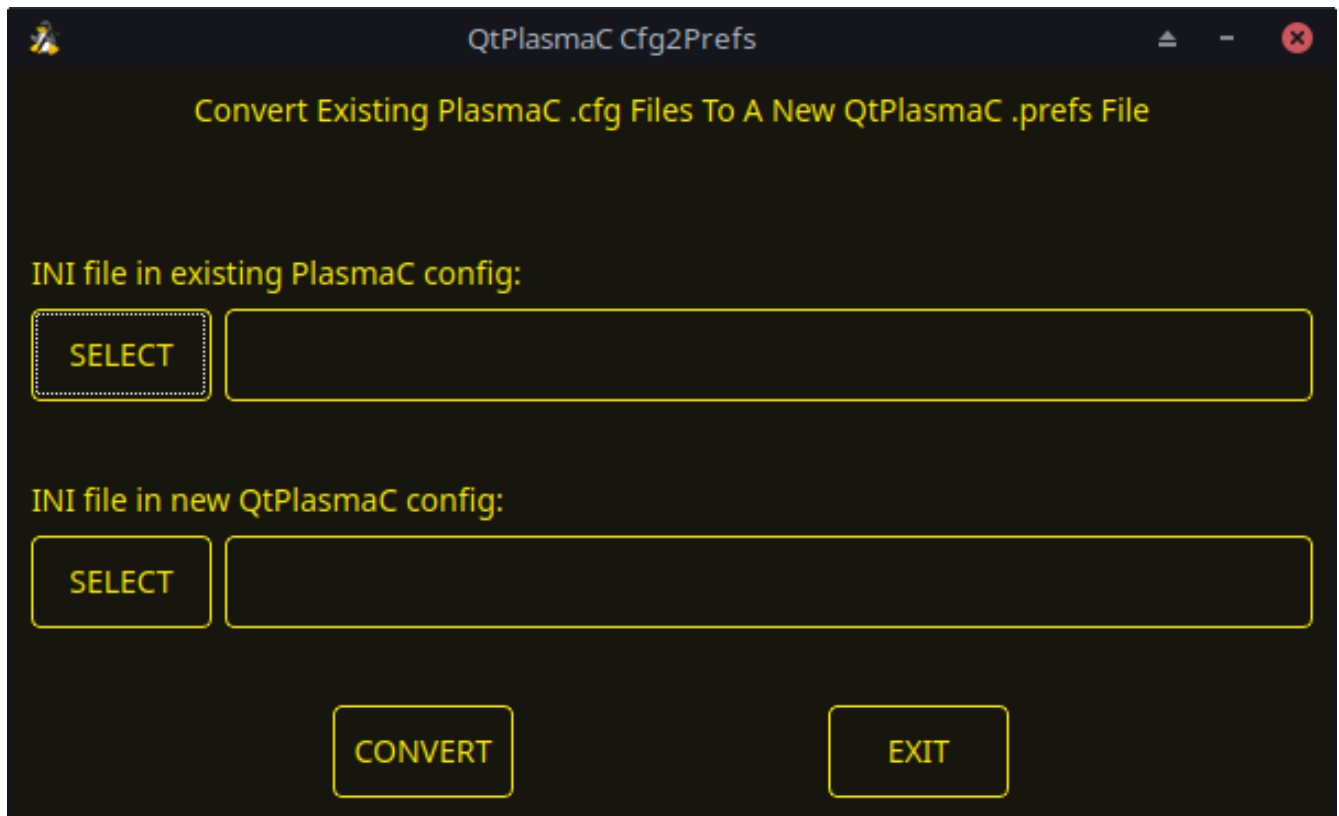
To run the cfg2prefs conversion program, use the following instructions:

For a package installation (Buildbot) enter the following line in a terminal window:

```
qtplasmac-cfg2prefs
```

For a run in place installation enter the following lines in terminal window:

```
source ~/linuxcnc-dev/scripts/rip-environment
qtplasmac-cfg2prefs
```



Select the .ini file of the old PlasmaC configuration, select the .ini file of the new QtPlasmaC configuration, then press **CONVERT**.

5.7.7 Other QtPlasmaC Setup Considerations

5.7.7.1 Lowpass Filter

The plasmac HAL component has a built in lowpass filter that if used is applied to the **plasmac.arc-voltage-in** input pin to filter any noise that could cause erroneous voltage readings. The lowpass filter should only be used after using Halscope to determine the required frequency and whether the amplitude of the noise is large enough to cause any issues. For most plasma machines lowpass is not required and should not be used unless it is required.

The HAL pin assigned to this filter is **plasmac.lowpass-frequency** and is set to 0 (disabled) by default. To apply a lowpass filter to the arc-voltage, the user would edit the following entry in the custom.hal file in the machine's configuration directory to add the appropriate cutoff frequency as measured in Hertz (Hz).

For example:

```
setp plasmac.lowpass-frequency 100
```

The above example would give a cutoff frequency of 100Hz.

5.7.7.2 Contact Bounce

Contact bounce from mechanical relays, switches, or external interference may cause some inconsistent behavior of the following switches:

- Float Switch
- Ohmic Probe
- Breakaway Switch
- Arc OK (for modes 1 & 2)

Due to the fact that the software is capable of sampling rates faster than the contact bounce period, it is possible that the software may see contact bounce as several changes in input states occurring in a very small time period, and incorrectly interpret this as a very quick on-off of the input. One method of mitigating contact bounce is to "debounce" the input. To summarize debounce, it requires the input state to be stable at the opposite state of the output state for consecutive delay periods before changing the state of the output.

Debounce delay periods can be changed by editing the appropriate debounce value in the custom.hal file in the <machine_name> config directory.

Each increment of delay adds one servo thread cycle to the debounce time. For example: given a servo thread period of 1000000 (measured in nano seconds), a debounce delay of 5 would equate to 5000000ns, or 5ms.

For the Float and Ohmic switches this equates to a 0.001mm (0.00004") increase in the probed height result.

It is recommended to keep the debounce values as low as possible while still achieving consistent results. Using [Halscope](#) to plot the inputs is a good way to establish the correct value.

For QtPlasmaC installations, debounce is achieved by using the HAL [dbounce component](#) which is a later alternative to the original debounce component. This new version allows for the loading and naming of individual debounce instances and is compatible with Twopass HAL file processing.

All four signals above have an individual debounce component so the debounce periods can be catered individually to each input. Any changes made to these values in the custom.hal file will not be overwritten by later updates of QtPlasmaC.

The default delay for all four inputs is five servo thread periods. In most cases this value will work quite well. If any of the inputs do not use mechanical switches, it may be possible to either reduce or remove the delay for those inputs.

If debounce is required for other equipment like home or limit switches etc. then more dbounce components may added in any of the HAL files without any regard to the signals listed here.

More information on contact bounce can be seen on page IX of [Finder Relays General Technical Information](#)

5.7.7.3 Contact Load

Mechanical relays and switches usually require a minimum current passing through the contacts for reliable operation. This current varies with the material that the contacts in the device are made from.

Depending on the specified minimum contact current and the current drawn by the input device there may be a need to provide a method to increase the current through the contacts.

Most relays using gold contacts will not require any additional current for reliable operation.

There are two different methods available to provide this minimum current if it is required:

1. A 0.1μF film capacitor placed across the contacts.
2. A 1200Ω 1W resistor across the load (see calculations below).

Schematics are shown at [contact load schematics](#).

More information on contact switching load can be seen on page III of [Finder Relays General Technical Information](#)

Calculations:

If using a Mesa card, the input resistance of a 7i96 is $4700\ \Omega$ (always consult the product manual associated with the revision being used as these values sometimes vary between revisions), giving a contact current of 5.1 mA assuming a supply voltage of 24 V ($I = V/R$).

As an example, the typical relay used in a Hypertherm Powermax 65 plasma cutter ([TE T77S1D10-24](#)) requires a minimum contact load of 100 mA @ 5 VDC which will dissipate 0.5 W ($P = I * V$). If using a 24 VDC power supply this would then equate to a minimum current of 20.8 mA. Because there is less current drawn by the Mesa input than is required by the relay there needs to be an increase in the current.

The resistance can be calculated using $R = V_s / (I_m - I_i)$ where:

- R = calculated resistance
- V_s = supply voltage
- I_m = minimum current required
- I_i = input current

Using a 7i96 with an input current of 5.1 mA gives a calculated value of $1529\ \Omega$. ($24 / (.0208 - .0051)$). This could then be rounded down to a commonly available $1500\ \Omega$ resistor giving a small safety margin.

The power dissipation can be calculated using $P = V_s^2 / R_s$ where:

- P = power
- V_s = supply voltage
- R_s = selected resistance

This gives a value of 0.38W. This could then be rounded up to 1W, giving a good safety margin. The final selection would be a $1500\ \Omega$ 1W resistor.

5.7.7.4 Desktop Launcher

If a link to the launch the configuration was not created when creating the config, the user could create a desktop launcher to the config by right clicking on the desktop and selecting Create Launcher or similar. This will bring up a dialog box to create a launcher. Give the icon a nice short name, enter anything for the command and click OK.

After the launcher appears on the desktop, right click on it and then edit it with the user's editor of choice. Edit the file so it looks similar to:

```
[Desktop Entry]
Comment=
Terminal=false
Name=LinuxCNC
Exec=sh -c "linuxcnc $HOME/linuxcnc/configs/<machine_name>/<machine_name>.ini"
Type=Application
Icon=/usr/share/pixmaps/linuxcncicon.png
```

If the user would like a terminal window to open behind the GUI window then change the Terminal line to:

```
Terminal=true
```

Displaying a terminal can be handy for error and information messages.

5.7.7.5 QtPlasmaC Files

After a successful QtPlasmaC installation, the following files are created in the configuration directory:

Filename	Function
<machine_name>.ini	A configuration file for the machine.
<machine_name>.hal	A HAL for the machine.
<machine_name>.prefs	A configuration file for QtPlasmaC specific parameters and preferences.
custom.hal	A HAL file for user customization.
custom_postgui.hal	A HAL file for user customization which is run after the GUI has initialized.
shutdown.hal	A HAL file which is run during the shutdown sequence.
tool.tbl	A tool table used to store offset information for additional tools (scribe, etc.) used by the QtPlasmaC configuration.
qtplasmac	A link to the directory containing common qtplasmac support files.
backup	A directory for backups of config files.

Note

<machine_name> is whatever name the user entered into the "Machine Name" field of the configuration wizard program

Note

Custom commands are allowed in custom.hal and the custom_postgui.hal files as they are not overwritten during updates.

After running a new configuration for the first time the following files will be created in the configuration directory:

Filename	Function
<machine_name>_material.cfg	A file for storing the material settings from the MATERIAL section of the PARAMETERS Tab .
qtvcp.prefs	A file containing the QtVCP preferences.
qtplasmac.qss	This file is used to store the stylesheet for the currently loaded session of QtPlasmaC.

Note

The configuration files (<machine_name>.ini and <machine_name>.hal) that are created by configuration wizard are notated to explain the requirements to aid in manual manipulation of these configurations. They may be edited with any text editor.

Note

The <machine_name>.prefs file is plain text and may be edited with any text editor.

5.7.7.6 INI File

QtPlasmaC has some specific <machine_name>.ini file variables as follows:

[FILTER] Section

These variables are mandatory.

```

PROGRAM_EXTENSION = .ngc,.nc,.tap G-code File (*.ngc, *.nc, *.tap)
ngc                = qtplasmac_gcode
nc                 = qtplasmac_gcode
tap                = qtplasmac_gcode

```

[RS274NGC] Section

These variables are mandatory.

```

RS274NGC_STARTUP_CODE = G21 G40 G49 G80 G90 G92.1 G94 G97 M52P1
SUBROUTINE_PATH       = ./:.././nc_files
USER_M_PATH           = ./:.././nc_files

```

Note

for a imperial config replace G21 above with G20.

Note

both the above paths show the minimum requirements.



Important

SEE [PATH TOLERANCE](#) FOR RS274NGC_STARTUP_CODE INFORMATION RELATED TO G64.

[HAL] Section

These variables are mandatory.

```

HALUI           = halui (required)
HALFILE         = <machine_name>.hal (the machine HAL file)
HALFILE         = plasmac.tcl (the standard QtPlasmaC HAL file )
HALFILE         = custom.hal (Users custom HAL commands)
POSTGUI_HALFILE = postgui_call_list.hal (required)
SHUTDOWN        = shutdown.hal (shutdown HAL commands)

```

Note

The user could place custom HAL commands in the custom.hal file as this file is not overwritten by QtPlasmaC updates.

[DISPLAY] Section

This variable is mandatory.

```

DISPLAY = qtvcp qtplasmac      (use 16:9 resolution)
        = qtvcp qtplasmac_9x16 (use 9:16 resolution)
        = qtvcp qtplasmac_4x3  (use 4:3 resolution)

```

There are multiple QtVCP options that are described here: [QtVCP INI Settings](#)

For example the following would start a 16:9 resolution QtPlasmaC screen in full screen mode:

```
DISPLAY = qtvcp -f qtplasmac
```

[TRAJ] Section

This variable is mandatory.

```
SPINDLES = 3
```

[AXIS_X] Section

These variables are mandatory.

```
MAX_VELOCITY      = double the value in the corresponding joint
MAX_ACCELERATION  = double the value in the corresponding joint
OFFSET_AV_RATIO   = 0.5
```

[AXIS_Y] Section

These variables are mandatory.

```
MAX_VELOCITY      = double the value in the corresponding joint
MAX_ACCELERATION  = double the value in the corresponding joint
OFFSET_AV_RATIO   = 0.5
```

[AXIS_Z] Section

These variables are mandatory.

```
MIN_LIMIT         = just below the top of the table's slats
MAX_VELOCITY      = double the value in the corresponding joint
MAX_ACCELERATION  = double the value in the corresponding joint
OFFSET_AV_RATIO   = 0.5
```

Note

QtPlasmaC uses the LinuxCNC External Offsets feature for all Z axis motion, and for moving the X and/or Y axis for a consumable change while paused. For more information on this feature, please read [External Axis Offsets](#) in the LinuxCNC documentation.

5.7.8 QtPlasmaC GUI Overview

The following sections will give a general overview of the QtPlasmaC layout.

5.7.8.1 Exiting QtPlasmaC

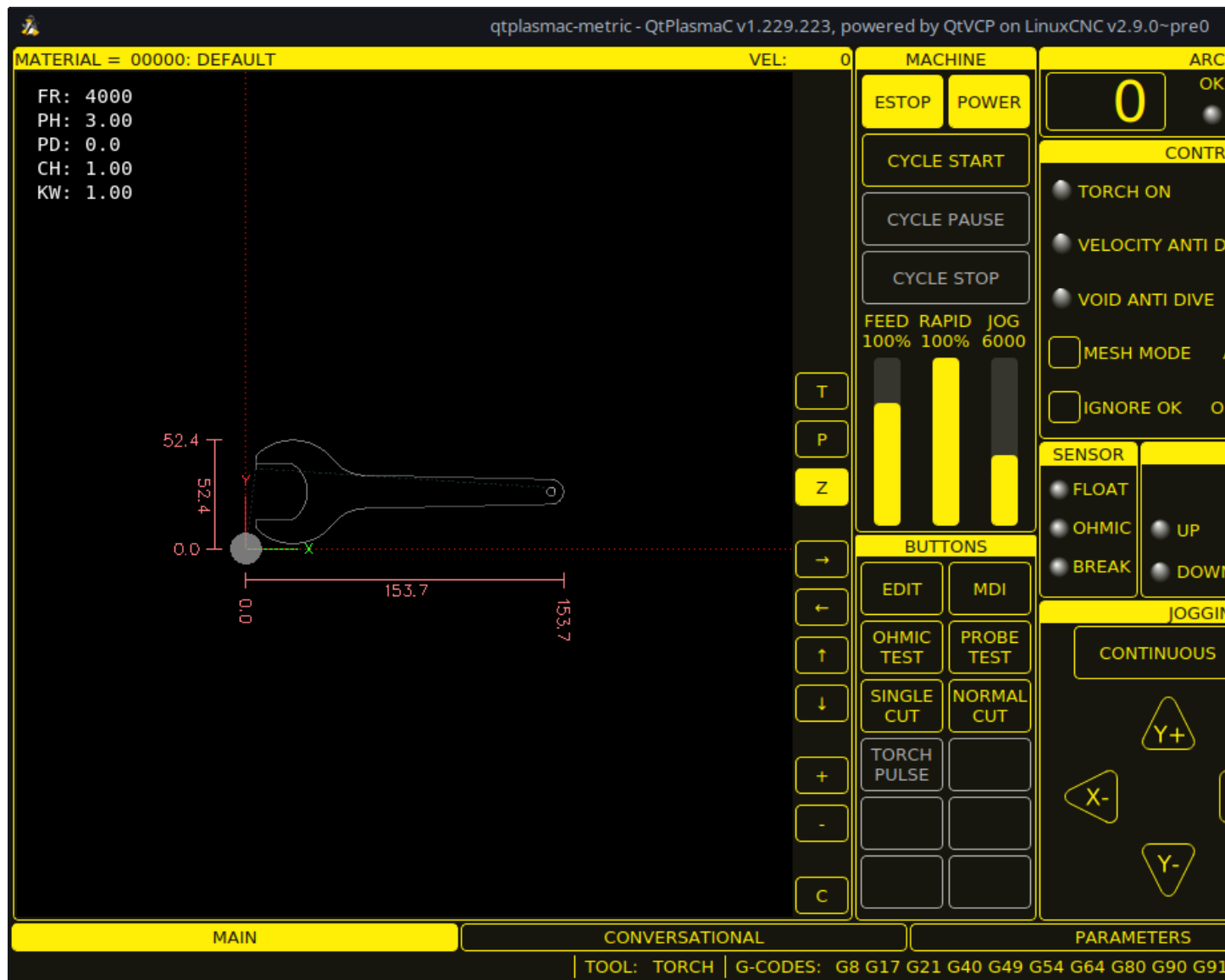
Exiting or shutting down QtPlasmaC is done by either:

1. Click the window shutdown button on the window title bar
2. Long press the **POWER** button on the MAIN Tab.

A shutdown warning can be displayed on every shutdown by checking the *Exit

5.7.8.2 MAIN Tab

Screenshot example of the QtPlasmaC **MAIN Tab** in **16:9** aspect ratio:



Some functions/features are only used for particular modes and are not displayed if they are not required by the chosen QtPlasmaC mode.

PREVIEW WINDOW

Name	Description
Material	The top header is clickable in this area to reveal a drop down menu. It is used to manually select the current material cut parameters. If there are no materials in the material file then only the default material will be displayed.
VEL:	This displays the actual cut feed rate the table is moving at.
FR:	If "View Material" is selected on the PARAMETERS Tab , this displays the currently selected material's Feed Rate.
PH:	If "View Material" is selected on the PARAMETERS Tab , this displays the currently selected material's Pierce Height.

Name	Description
PD:	If "View Material" is selected on the PARAMETERS Tab , this displays the currently selected material's Pierce Delay.
CH:	If "View Material" is selected on the PARAMETERS Tab , this displays the currently selected material's Cut Height.
CA:	If "View Material" is selected on the PARAMETERS Tab , and RS485 communications are enabled, this displays the currently selected material's Cut Amperage.
T	This button changes the preview to a top down full table view.
P	This button changes the preview to an isometric view.
Z	This button changes the preview to a top down view.
→	This button pans the preview right.
←	This button pans the preview left.
↑	This button pans the preview up.
↓	This button pans the preview down.
+	This button zooms the preview .
-	This button zooms the preview .
C	This button clears the live plot.

MACHINE

Name	Description
ESTOP	<p>If ESTOP_TYPE = 0 in the <machine_name>.prefs file, this button becomes an indicator of the hardware ESTOP's status only.</p> <p>If ESTOP_TYPE = 1 in the <machine_name>.prefs file, this button will not be visible.</p> <p>If ESTOP_TYPE = 2 in the <machine_name>.prefs file, this button will act as a GUI ESTOP.</p> <p>If ESTOP_TYPE is omitted from the <machine_name>.prefs file, this button will default to being an indicator of the hardware ESTOP's status only.</p>
POWER	<p>This button turns the GUI on and allows QtPlasmaC/LinuxCNC to control the hardware.</p> <p>Pressing and holding the POWER button for longer than two seconds will bring up a dialog to exit the QtPlasmaC application.</p>
CYCLE START	This button starts the cycle for any loaded G-code file.
CYCLE PAUSE	<p>This button pauses the cycle for any loaded G-code file.</p> <p>If a cycle is paused, this button will display CYCLE RESUME and flash.</p> <p>Pressing CYCLE RESUME will resume the cycle.</p>
CYCLE STOP	<p>This button stops any actively running or paused cycle.</p> <p>This includes:</p> <ul style="list-style-type: none"> - G-code Programs - Torch pulse if the pulse was started during CYCLE PAUSE (this will cancel the paused G-code program execution as well) - Probe Test - Framing - Manual Cut
FEED	<p>This slider overrides the feed rate for all feed moves.</p> <p>Any value other than 100% will cause the label to flash.</p> <p>Clicking the label will return the slider to 100%.</p>
RAPID	<p>This slider overrides the rapid rate for all rapid moves.</p> <p>Any value other than 100% will cause the label to flash.</p> <p>Clicking the label will return the slider to 100%.</p>
JOG	<p>This slider sets the jog rate.</p> <p>Clicking the label will return the slider to the default linear velocity as set in the <machine_name>.ini file.</p>

BUTTONS

The Button Panel contains buttons useful for the operation of the machine.

The **EDIT** and **MDI** buttons are permanent, all other buttons are user programmable in the <machine_name>.prefs file.

See [custom user buttons](#) for detailed information on custom user buttons.

Name	Description
EDIT	This button opens a G-code editor for the currently loaded program.
MDI	This button places QtPlasmaC into Manual Data Input (MDI) mode which will display the MDI HISTORY and an entry box over top of the G-code window. Once pressed, this button will display "MDI CLOSE". Pressing MDI CLOSE will close the MDI. Please see the MDI section for additional MDI information.
OHMIC TEST	This button will enable the Ohmic Probe Enable output signal and if the Ohmic Probe input is sensed, the LED indicator in the SENSOR Panel will light. The main purpose of this is to allow a quick test for a shorted torch tip.
PROBE TEST	This button will initiate a Probe Test .
SINGLE CUT	This button will show the dialog box to start an automatic Single Cut .
NORMAL CUT	This button will toggle between Cut Types (NORMAL CUT and PIERCE ONLY).
TORCH PULSE	This button will initiate a Torch Pulse .

ARC

Name	Modes	Description
Arc Voltage	0, 1	Displays the actual arc voltage.
OK	0, 1, 2	Indicates the status of the Arc OK signal.
+	0, 1	Each press of this button will raise the target voltage by the THC Threshold voltage (The distance changed will be Height Per Volt * THC Threshold voltage).
-	0, 1	Each press of this button will lower the target voltage by the THC Threshold voltage (The distance changed will be Height Per Volt * THC Threshold voltage).
OVERRIDE	0, 1	Clicking this label will return any voltage override to 0.00.

CONTROL

Name	Modes	Description
TORCH ON	0, 1, 2	Indicates the status of the Torch On output signal.
TORCH ON ENABLE	0, 1, 2	This box toggles between Enabling and Disabling the torch. This box defaults to unfilled (disabled) when QtPlasmaC is first run. This box must be filled to change it to "Torch Enabled" before material cutting can commence. If this box is not filled, then running a loaded program will cause the machine to run the cycle without firing the torch. This is sometimes referred to as a "dry run".

Name	Modes	Description
VELOCITY ANTI DIVE	0, 1, 2	Indicates that the THC is locked at the current height due to the cut velocity falling below the Velocity Anti Dive (VAD) Threshold percentage set on the PARAMETERS Tab .
VELOCITY ANTI DIVE ENABLE	0, 1, 2	This box toggles between Enabling and Disabling VELOCITY ANTI DIVE.
VOID ANTI DIVE	0, 1	Indicates that the THC is locked due to a void being sensed.
VOID ANTI DIVE ENABLE	0, 1	This box toggles between Enabling and Disabling VOID ANTI DIVE.
MESH MODE	0, 1, 2	This box will enable or disable Mesh Mode for the cutting of expanded metal. This check box may be enabled or disabled at any time during normal cutting. Mesh mode: <ul style="list-style-type: none"> - Will require an Arc OK signal to start machine motion. - Will disable the THC. - Will not stop machine motion if the Arc OK signal is lost. - Will automatically select CPA mode if PowerMax communications are being used. For more information see Mesh Mode (expanded metal) .
AUTO VOLTS	0, 1	This box will enable or disable Auto Volts .
IGNORE OK	0, 1, 2	This box will determine if QtPlasmaC ignores the Arc OK signal. This check box may be enabled or disabled at any time during normal cutting. Additionally this mode may be enabled or disabled via proper M codes in a running program. Ignore Arc OK mode: <ul style="list-style-type: none"> - Will not require an Arc OK signal be received before starting machine motion after the "Torch On" signal is given. - Will disable the THC. - Will not stop machine motion if the Arc OK signal is lost. For more information see Ignore Arc Ok .
OHMIC PROBE	0, 1, 2	This box enables or disables the ohmic probe input. If the Ohmic Probe input is disabled, the Ohmic Probe LED will still show the status of the probe input, but the Ohmic Probe results will be ignored.
RS485	0, 1, 2	This box will enable or disable the communications to a PowerMax. This button is only visible if a PM_PORT is configured in the [POWERMAX] section of the <machine_name>.prefs file.
Status	0, 1, 2	When PowerMax communications are enabled, this will display one of the following: CONNECTING, CONNECTED, COMMS ERROR, or a Fault Code. For more information, see the PowerMax Communications section.

SENSOR

Name	Description
FLOAT	Indicates that the float switch is activated.
OHMIC	Indicates that the probe has sensed the material.
BREAK	Indicates that the torch breakaway sensor is activated.

THC

Name	Description
ENABLE	This box determines whether the THC will be enabled or disabled during a cut.
ENABLED	This LED indicates whether the THC is enabled or disabled.
ACTIVE	This LED indicates that the THC is actively controlling the Z axis.
UP	This LED indicates that the THC is commanding the Z axis to raise.
DOWN	This LED indicates that the THC is commanding the Z axis to lower.

JOGGING

Note

During Paused Motion, this section will become [CUT RECOVERY](#)

Name	Description
CONTINUOUS	This drop down button will change the jog increment. Options are determined by the values in the [DISPLAY] section of the <machine_name>.ini file and begin with the label "INCREMENTS =".
FAST	This button will toggle between FAST which is the default linear velocity in the <machine_name>.ini file or SLOW which is 10% of the default value.
Y+	This button moves the Y axis in the positive direction.
Y-	This button moves the Y axis in the negative direction.
X+	This button moves the X axis in the positive direction.
X-	This button moves the X axis in the negative direction.
Z+	This button moves the Z axis in the positive direction.
Z-	This button moves the Z axis in the negative direction.

CUT RECOVERY

Note

During Paused Motion, this section will be shown on top of the JOGGING panel. The following section will cover each button encountered in this panel. Please see [CUT RECOVERY](#) for a detailed description of the cut recovery functionality.

Name	Description
PAUSED MOTION FEED SLIDER	In the event of a paused program, this interface allows X/Y motion to follow the programmed path in the reverse or forward direction. This slider's range is from 1%-100% of the Cut Feed Rate for the currently selected material.
FEED	This displays the paused motion feed rate.
REV	In the event of a paused program, this button will move the machine in reverse along the programmed path until it reaches the last M3 command that was either executed or that QtPlasmaC was attempting to execute before the program became paused.
FWD	In the event of a paused program, this button will move the machine forward along the programmed path indefinitely until the program's end, skipping over M3 commands.
CANCEL MOVE	This button will cancel any Cut Recovery movement that was made, and return the torch to the position the Cut Recovery movement was initiated. Note that if FWD or REV were used to move the torch, CANCEL will not return to the position of the torch when the pause occurred.

Name	Description
MOVE x.xxx	This displays the amount of travel that will be incurred with each press of an arrow key, in the direction the arrow key was pressed. This value displayed below MOVE represents the Kerf Width of the currently selected material.
DIRECTIONAL ARROWS	These buttons will move the torch in the direction indicated by a distance of one Kerf Width (of the currently selected material) per press.

G-CODE WINDOW

Name	Description
CLEAR	This button will clear the currently opened program. The torch (T0) will be selected if it was not the active tool.
OPEN	This button will open a FILE OPEN panel over the PREVIEW WINDOW.
RELOAD	This button will reload the currently loaded G-code File.

DRO

Name	Description
HOME ALL	This button will home all of the axes in the order set by HOME SEQUENCE in the <machine_name>.ini file.
WCS G54	This drop down button will change the current work offset.
CAMERA	This button will display a CAMVIEW panel on top of the PREVIEW WINDOW and will allow the user to set an origin with or without rotation. See the CAMERA section for detailed instructions.
LASER	This button will allow the user to use a laser to set an origin with or without rotation. See the LASER section for detailed instructions.
X0 Y0	This button will set the current position to X0 Y0.
HOME [AXIS]	This button will home the corresponding axis.
0 [AXIS]	This drop down button will display the following options: Zero - zeros the axis. Set - launches a dialog box to manually input the axis' coordinate. Divide By 2 - divides the currently displayed coordinate in the DRO by two. Set To Last - sets the axis to the previously set coordinate.

5.7.8.3 Preview Views

The QtPlasmaC preview screen has the ability to be switched between different views and displays, as well as zooming in and out, and panning horizontally and vertically.

When QtPlasmaC is first started, the Z (top down) view will be selected as the default view for a loaded G-code file, but the full table view will be displayed.

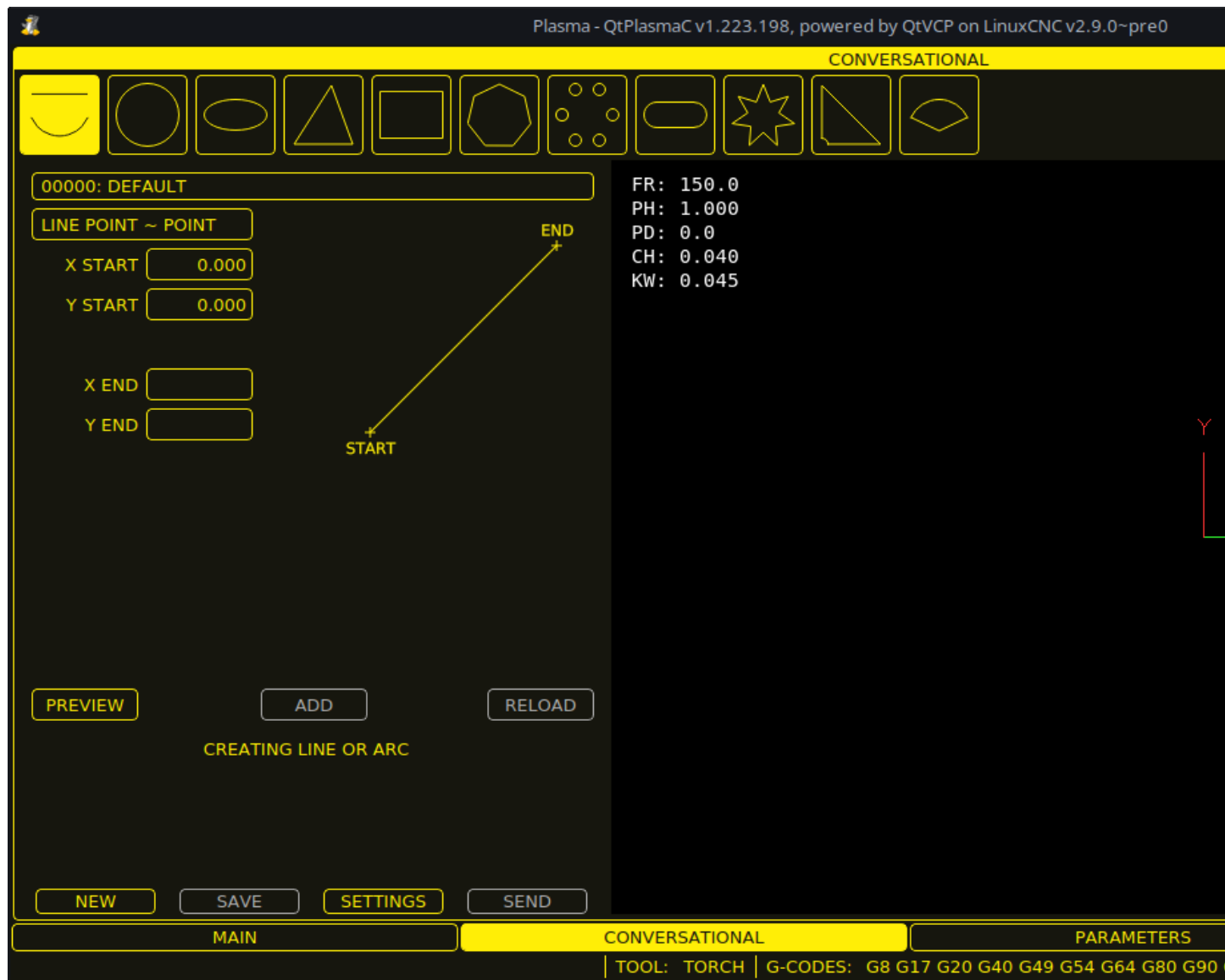
When a G-code file is loaded, the display will change to the selected view.

Whenever there is no G-code file loaded, the full table will automatically be displayed irrespective of which view is currently selected (the highlighted button representing the currently selected view will not change).

If a full table is displayed due to no G-code file being loaded and the user wishes to change the view orientation, then pressing either Z or P will change the display to the newly selected view. If the user then wishes to display the full table while maintaining the currently selected view as the default view for a loaded G-code file, then pressing CLEAR will achieve this and allow the selected view orientation to prevail the next time a G-code file is loaded.

5.7.8.4 CONVERSATIONAL Tab

Screenshot example of the QtPlasmaC [CONVERSATIONAL Tab](#) in **16:9** aspect ratio:



The [CONVERSATIONAL Tab](#) enables the user to quickly program various simple shapes for quick cutting without the need for CAM software.

See [Conversational Shape Library](#) for detailed information on the Conversational feature.

It is possible to disable this tab so the conversational feature cannot be used by an operator. This may be achieved either by wiring the pin to a physical key-switch or similar or it may also be set in a HAL file using the following command:

```
setp qtplasmac.conv_disable 1
```

5.7.8.5 PARAMETERS Tab

Screenshot example of the QtPlasmaC [PARAMETERS Tab](#) in **16:9** aspect ratio:



Some functions/features are only used for particular modes and are not displayed if they are not required by the chosen QtPlasmaC mode.

Due to space constraints, the 4x3 GUI's [PARAMETERS Tab](#) will be spread across two tabs, PARAMETERS and SETTINGS.

This tab is used to display configuration parameters that are modified infrequently.

It is possible to disable this tab so machine settings cannot be modified by unauthorized personnel. This may be achieved either by wiring the pin to a physical key-switch or similar or it may also be set in a HAL file using the following command:

```
setp qtplasmac.param_disable 1
```

CONFIGURATION - ARC

Name	Modes	Description
Start Fail Timer	0, 1, 2	This sets the amount of time (in seconds) QtPlasmaC will wait between commanding a "Torch On" and receiving an Arc OK signal before timing out and displaying an error message.

Name	Modes	Description
Max Starts	0, 1, 2	This sets the number of times QtPlasmaC will attempt to start the arc.
Retry Delay	0, 1, 2	This sets the time (in seconds) between an arc failure and another arc start attempt.
Voltage Scale	0, 1	This sets the arc voltage input scale and is used to display the correct arc voltage. For initial setup, see Calibration Values .
Voltage Offset	0, 1	This sets the arc voltage offset and is used to display zero volts when there is zero arc voltage input. For initial setup, see Calibration Values .
Height Per Volt	0, 1, 2	This sets the distance the torch would need to move to change the arc voltage by one volt. Used for manual height manipulation only.
OK High Volts	0	This sets the voltage threshold below which Arc OK signal is valid.
OK Low Volts	0	This sets the voltage threshold above which the Arc OK signal is valid.

Note

When setting the OK Low Volts and OK High Volts in Mode 0, the cut voltage of a stable arc must be greater than the OK Low Volts value but lower than the OK High Volts value for QtPlasmaC to receive a valid Arc OK signal. To further clarify, to have a valid Arc OK, the arc voltage must fall between the two limits.

CONFIGURATION - PROBING

Name	Description
Float Travel	This sets the amount of travel the float switch moves before completing the float switch circuit. This distance can be measured by using the Probe Test button, and the method described in Initial Setup .
Probe Speed	This sets the speed at which the torch will probe to find the material after it moves to the Probe Height.
Probe Height	This sets the height above the Z axis minimum limit that Probe Speed begins. Refer to the Heights Diagram diagram for a visual representation.
Ohmic Offset	This sets the distance above the material the torch will should go after a successful ohmic probe. It is mainly used to compensate for high probing speeds.
Ohmic Retries	This sets the number of times QtPlasmaC will retry a failed ohmic probe before falling back to the float switch for material detection.
Skip IHS	This sets the distance threshold used to determine if an Initial Height Sense (probe) can be skipped for the current cut, see IHS Skip .

Note

If the amount of time between the torch contacting the material and when the torch moves up and comes to rest at the Pierce Height seems excessive, see [the probing section](#) for a possible solution.

CONFIGURATION - SAFETY

Name	Description
Safe Height	This sets the height above the material that the torch will retract to before executing rapid moves. If set to Zero then Z axis maximum height will be used for the safe height. Refer to the Heights Diagram diagram for a visual representation.

CONFIGURATION - SCRIBING

Name	Description
Arm Delay	This sets the delay (in seconds) from the time the scribe command is received to the activation of the scribe. This allows the scribe to reach surface of the material before activating the scribe.
On Delay	This sets the delay (in seconds) to allow the scribe mechanism to start before beginning motion.

CONFIGURATION - SPOTTING

Name	Description
Threshold	This sets the arc voltage at which the delay timer will begin. 0V starts the delay when the torch on signal is activated.
Time On	This sets the length of time (in milliseconds) the torch is on after threshold voltage is reached.

CONFIGURATION - MOTION

Name	Description
Max. Speed	Displays the maximum velocity the Z axis is capable of (this is controlled by the <machine_name>.ini file).
Setup Speed	The Z axis velocity for setup moves (movements to Probe Height, Pierce Height, Cut Height, etc.).

Note

Setup Speed has no effect on THC speed which is capable of the velocity displayed in the Max. Speed field.

CONFIGURATION - THC

Name	Modes	Description
Delay	0, 1, 2	This sets the delay (in seconds) measured from the time the Arc OK signal is received until Torch Height Controller (THC) activates.
Threshold	0, 1, 2	This sets the voltage variation allowed from the target voltage before for THC makes movements to correct the torch height.
Speed (PID-P)	0, 1	This sets the Proportional gain for the THC PID loop. This roughly equates to how quickly the THC attempts to correct changes in height.
VAD Threshold	0, 1	(Velocity Anti Dive) This sets the percentage of the current cut feed rate the machine can slow to before locking the THC to prevent torch dive.
Void Override	0, 1	This sets the size of the change in cut voltage necessary to lock the THC to prevent torch dive (higher values need greater voltage change to lock THC).

Name	Modes	Description
PID-I	0, 1	This sets the Integral gain for the THC PID loop. Integral gain is associated with the sum of errors in the system over time and is not always needed.
PID-D	0, 1	This sets the Derivative gain for the THC PID loop. Derivative gain works to dampen the system and reduce over correction oscillations and is not always needed.

Note

PID loop tuning is a complicated process and is outside the scope of this User Guide. There are many sources of information available to assist with understanding and tuning PID loops. If the THC is not making corrections fast enough, it is recommended to increase the P gain in small increments until the system operates favorably. Large P gain adjustments can result in over correction and oscillations.

SAVE & RELOAD Buttons

The **SAVE** button will save the currently displayed parameters to the <machine_name>.prefs file.

The **RELOAD** button will reload all the parameters from the <machine_name>.prefs file.

MATERIAL

This section shows the parameters which are active for the current cut.

Name	Description
Material	The top drop down menu is used to manually select the current material cut parameters. If there are no materials in the material file then only the default material will be displayed.
Kerf Width	This sets the kerf width for the currently selected material. Refer to the Heights Diagram diagram for a visual representation.
Pierce Height	This sets the pierce height for the currently selected material. Refer to the Heights Diagram diagram for a visual representation.
Pierce Delay	This sets the pierce delay (in seconds) for the currently selected material.
Cut Height	This sets the cut height for the currently selected material. Refer to the Heights Diagram diagram for a visual representation.
Cut Feed Rate	This sets the cut feed rate for the currently selected material.
Cut Amps	This sets the cut amperage for the currently selected material. This is a visual indicator to the operator only, unless PowerMax communications are being used.
Cut Volts	This sets the cut voltage for the currently selected material.
Puddle Height	Expressed as a percentage of Pierce Height, this sets the Puddle Jump height for the currently selected material. Typically used for thicker materials, Puddle Jump allows the torch to have an intermediate step between Pierce Height and Cut Height. If set, the torch will proceed from Pierce Height to P-Jump Height for a period of time (P-Jump Delay) before proceeding to Cut Height to effectively "jump" over the molten puddle. Refer to the Heights Diagram diagram for a visual representation.
Puddle Delay	This sets the amount of time (in seconds) the torch will stay at the P-Jump Height before proceeding to Cut Height.
Pause At End	This sets the amount of time (in seconds) the torch will stay on at the end of the cut before proceeding with the M5 command to turn off and raise the torch. For more information see Pause At End Of Cut .
Gas Pressure	This sets the gas pressure for the currently selected material. This setting is only valid if PowerMax communications are being used. 0 = Use the PowerMax's automatic pressure mode.

Name	Description
Cut Mode	This sets the cut mode for the currently selected material. This setting is only valid if PowerMax communications are being used. 1 = Normal 2 = CPA (Constant Pilot Arc) 3 = Gouge/Mark

Note

SEE THE [THICK MATERIALS](#) SECTION FOR MORE INFORMATION ON PUDDLE JUMP

SAVE, RELOAD, NEW, & DELETE Buttons

The **SAVE** button will save the current material set to the <machine_name>_material.cfg file.

The **RELOAD** button will reload the material set from the <machine_name>_material.cfg file.

The **NEW** button will allow a new material to be added to the material file. The user will be prompted for a material number and a material name, all other parameters will be read from the currently selected material. Once entered, QtPlasmaC will reload the material file and display the new material. The Cut Parameters for the new material will then need to be adjusted and saved.

The **DELETE** this button is used to delete a material. After pressing it, the user will be prompted for a material number to be deleted, and prompted again to ensure the user is sure. After deletion, the material file will be reloaded and the drop down list will display the default material.

GUI SETTINGS

This section shows parameters that effect the GUI appearance and GUI behaviors.

To return any of the color changes to their default values, see the [Returning To The Default Styling](#) section.

Name	Description
Foreground	This button allows the user to change the color of the GUI Foreground.
Highlight	This button allows the user to change the color of the GUI Highlight.
LED	This button allows the user to change the color of the GUI LED.
Background	This button allows the user to change the color of the GUI Background.
Alt Background	This button allows the user to change the color of the GUI Alternate Background.
Frames	This button allows the user to change the color of the GUI Frames.
Estop	This button allows the user to change the color of the GUI Estop.
Disabled	This button allows the user to change the color of the GUI's Disabled features.
Preview	This button allows the user to change the color of the GUI Preview Window Background.
Soft Keyboard	This radio button allows the user to enable or disable the soft touchscreen keyboard. If the "onboard" virtual keyboard is installed then the custom layouts will be enabled .
KB Shortcuts	This radio button allows the user to enable or disable Keyboard Shortcuts within the GUI (such as keyboard jogging). In addition to the standard jog keys, a list of the additional shortcuts is available in the keyboard shortcuts section.
View Material	This radio button allows the user to enable or disable the addition of a visual reference showing key material cut settings to the Preview Windows of the MAIN and CONVERSATIONAL tabs. Examples are: Feed Rate, Pierce Height, Pierce Delay, and Cut Height. Cut Amps will be shown if PowerMax communications are enabled.

Name	Description
Exit Warning	This radio button allows the user to enable or disable whether a warning will always be displayed during shutdown.
Optional Stop	This radio button allows the user to enable or disable whether or not a running program will pause at an M1 command.
Run From Line	This radio button allows the user to enable or disable Run From Line . If enabled, the user can click on a line of G-code and have the program start from that line.
Override Limits	This radio button allows the user to temporarily Override the input from a Limit Switch in the event the limit switch becomes tripped during operation. This button can only be clicked when a limit switch is tripped.
Override Jog	This radio button will also allow jogging while jogging is inhibited due to a float switch, breakaway switch, or ohmic probe activation. This button can only be clicked when a jog is inhibited.
Optional Block	This radio button allows the user to enable or disable whether or not lines starting with "/" will be skipped if present in a running program.
Grid Size	This allows a user to change the size of the grid in the Preview Window on the MAIN Tab . Grid size of 0.0 will disable the grid.
Cone Size	This allows a user to change the size of the cone (which represents the current tool) in the Preview Window on the MAIN Tab .
Table Zoom	This allows a user to change the default zoom level for the top down full table view in the Preview Window on the MAIN Tab .

UTILITIES

Some standard LinuxCNC utilities are provided as an aid in the diagnosis of issues that may arise:

- [Halshow](#)
- [Halscope](#)
- [Halmeter](#)
- [Calibration](#)
- [Status](#)

In addition the following two QtPlasmaC specific utilities are provided:

The **SET OFFSETS** button is used if the table has a laser or camera for sheet alignment, a scribe, or uses offset probing. The required offsets for these peripherals need to be applied by following the procedure described in [Peripheral Offsets](#).

The **BACKUP CONFIG** button will create a complete machine configuration backup for archiving or to aid in fault diagnosis. A compressed backup of the machine configuration will be saved in the user's Linux home directory. The file name will be <machine_name>_<version>_<date>_<time>.tar.gz where <machine_name> is the machine name entered in the configuration wizard, <version> is the current QtPlasmaC version the user is on, <date> is the current date (YY-MM-DD), and <time> is the current time (HH-MM-SS).

Prior to the backup being made, the machine log will be saved to a file in the configuration directory named machine_log_<date>_<time>.txt where <date> and <time> are as described above. This file along with up to five previous machine logs will also be included in the backup.

These files are not required by QtPlasmaC and are safe to delete at any time.

5.7.8.6 STATISTICS Tab

The [STATISTICS Tab](#) provides statistics to allow for the tracking of consumable wear and job run times.

These statistics are shown for the current job as well as the running total.

Previous job statistics are reset once the next program is run.

The total values may be reset either individually by clicking the corresponding "RESET" button, or they may all be reset together by clicking "RESET ALL".

The **RS485 PMX STATISTICS** panel will only be displayed if the user has Hypertherm PowerMax communications and a valid RS485 connection to the PowerMax is established. This panel will show the **ARC ON TIME** for the PowerMax in hh:mm:ss format.

The **MACHINE LOG** is also displayed on the [STATISTICS Tab](#), this log will display any errors and/or important information that occurs during the current LinuxCNC session. If the user makes a backup of the configuration from the [PARAMETERS Tab](#) then the machine log is also included in the backup.

aaa - QtPlasmaC v1.0.39, powered by QtVCP on LinuxCNC v2.9.0~pre0

STATISTICS				RS485 PMX STATISTICS		MACHINE LOG
	ITEM	JOB	TOTAL	ITEM	TOTAL	
RESET	PIERCE COUNT	0	159	ARC ON TIME	0:00:00	--- Qtvcp Screen Started on: Sat, Jun 10:15:39 --- Sat12 10:15 Tool 0: No Tool
RESET	CUT LENGTH (Metres)	0.00	15.46			
RESET	CUT TIME	0.00	0:11:18			
RESET	TORCH ON TIME	0.00	0:12:02			
RESET	PROGRAM RUN TIME	0.00	6:00:24			
RESET	RAPID TIME	0.00	0:01:35			
RESET	PROBE TIME	0.00	0:06:05			
RESET ALL						

MAIN CONVERSATIONAL PARAMETERS

G-Codes: G8 G17 G21 G40 G49 G54 G64 G80 G90

5.7.9 Using QtPlasmaC

Once QtPlasmaC is successfully installed, no Z axis motion is required to be part of the G-code cut program. In fact, if any Z axis references are present in the cut program, the standard QtPlasmaC configuration will remove them during the program loading process.

For reliable use of QtPlasmaC the user should **NOT** use any Z axis offsets other than the coordinate system offsets (G54-G59.3).

QtPlasmaC will automatically add a line of G-code to move the Z axis to the correct height at the beginning of every G-code program.

Version Information - QtPlasmaC will display versioning information in the title of the main window. The information will be displayed as followed "QtPlasmaC vN.XXX.YYY - powered by QtVCP on LinuxCNC vZ.Z.Z" where N is the version of QtPlasmaC, XXX is the version of the HAL component (PlasmaC.comp), YYY is the GUI version, and Z.Z.Z is the version of LinuxCNC.

5.7.9.1 Units Systems

All settings and parameters in QtPlasmaC are required to be in the same units as specified in the ini file, being either metric or imperial.

If the user is attempting to run a G-code file that is in the "other" units system then all parameters including the material file parameters are still required to be in the native machines units. Any further conversions necessary to run the G-code file will be handled automatically by the G-code filter program.

For example: if a user had a metric machine and wished to run a G-code file that was set up to cut 1/4" thick material using imperial units (inch - G20) then the user with the metric machine would need to ensure that either the material number in the G-code file was set to the corresponding metric material to be cut, or that a new material is created with the correct metric parameters for the metric material to be cut. If the metric user wanted to cut the G-code file using imperial material, then the new material parameters would need to be converted from imperial units to metric when they are entered.

5.7.9.2 Preamble and Postamble Codes

The following stanzas are the minimum recommended codes to include in the preamble and postamble of any G-code file to be run by QtPlasmaC:

Metric:

```
G21 G40 G49 G64p0.1 G80 G90 G92.1 G94 G97
```

Imperial:

```
G20 G40 G49 G64p0.004 G80 G90 G92.1 G94 G97
```

A detailed explanation of each G-code can be found in the docs [here](#).

Note that throughout this user guide there are several additional recommendations for codes that are prudent to add to both the preamble and postamble depending on the features the user wishes to utilize.

5.7.9.3 Mandatory Codes

Aside from the preamble code, postamble code, and X/Y motion code, the only mandatory G-code syntax for QtPlasmaC to run a G-code program using a torch for cutting is **M3 \$0 S1** to begin a cut and **M5 \$0** to end a cut.

For backwards compatibility it is permissible to use **M3 S1** in lieu of **M3 \$0 S1** to begin a cutting job and **M5** in lieu of **M5 \$0** to end a cutting job. Note that this applies to cutting jobs only, for scribe and spotting jobs the **\$n** tool identifier is mandatory.

5.7.9.4 Coordinates

See [recommended Z axis](#) settings.

Each time LinuxCNC (QtPlasmaC) is started Joint homing is required. This allows LinuxCNC (QtPlasmaC) to establish the known coordinates of each axis and set the soft limits to the values specified in the <machine_name>.ini file in order to prevent the machine from crashing into a hard stop during normal use.

If the machine does not have home switches then the user needs to ensure that all axes are at the home coordinates specified in the <machine_name>.ini file before homing.

If the machine has home switches then it will move to the specified home coordinates when the Joints are homed.

Depending on the machine's configuration there will either be a **Home All** button or each axis will need to be homed individually. Use the appropriate button/buttons to home the machine.

As mentioned in the [Initial Setup](#) section, it is recommended that the first time QtPlasmaC is used that the user ensure there is nothing below the torch then jog the Z axis down until it stops at the Z axis MINIMUM LIMIT then click the 0 next to the Z axis DRO to **Touch Off** with the Z axis selected to set the Z axis at zero offset. This should not need to be done again.

If the user intends to place the material in the exact same place on the table every time, the user could jog the X and Y axes to the machine to the corresponding X0 Y0 position as established by the CAM software and then **Touch Off** both axes with a zero offset.

If the user intends to place the material randomly on the table then the user must **Touch Off** the X and Y axes at the appropriate position before starting the program.

5.7.9.5 Cut Feed Rate

QtPlasmaC is able to read a material file to load all the required cut parameters. To enable to G-code file to use the cut feed rate setting from the cut parameters use the following code in the G-code file:

```
F#<_hal[plasmac.cut-feed-rate]>
```

It is possible to use the standard G-code **F** word to set the cut feed rate as follows:

```
F 1000
```

If the **F** word is used and the **F** word value does not match the cut feed rate of the selected material then a warning dialog will indicate this during loading of the G-code file.

5.7.9.6 Material File

Material handling uses a material file that was created for the machine configuration when the configuration wizard was ran and allows the user to conveniently store known material settings for easy recall either manually or automatically via G-code. The resulting [material file](#) is named <machine_name>_material.cfg.

QtPlasmaC does not require the use of a material file. Instead, the user could change the cut parameters manually from the MATERIAL section of the [PARAMETERS Tab](#). It is also not required to use the automatic material changes. If the user does not wish to use this feature they can simply omit the material change codes from the G-code file.

It is also possible to not use the material file and [automatically load materials](#) from within the G-code file.

Material numbers in the materials file do not need to be consecutive nor do they need to be in numerical order.

The following variables are mandatory and an error message will appear if any are not found when the material file is loaded.

- PIERCE_HEIGHT
- PIERCE_DELAY
- CUT_HEIGHT
- CUT_SPEED

The following variables are optional. If they are not detected or have no value assigned, they will be assigned a value of 0 and no error message will appear.

- NAME
- KERF_WIDTH
- THC
- PUDDLE_JUMP_HEIGHT
- PUDDLE_JUMP_DELAY
- CUT_AMPS
- CUT_VOLTS
- PAUSE_AT_END
- GAS_PRESSURE
- CUT_MODE

Note

Material numbers 1000000 and above are reserved for temporary materials.



Warning

It is the responsibility of the operator to ensure that the variables are included if they are a requirement for the G-code to be run.

The material file uses the following format:

```
[MATERIAL_NUMBER_1]
NAME           = name
KERF_WIDTH     = value
THC            = value (0 = off, 1 = on)
PIERCE_HEIGHT  = value
PIERCE_DELAY   = value
PUDDLE_JUMP_HEIGHT = value
PUDDLE_JUMP_DELAY = value
CUT_HEIGHT     = value
CUT_SPEED      = value
CUT_AMPS       = value (for info only unless PowerMax communications is enabled)
CUT_VOLTS      = value (modes 0 & 1 only, if not using auto voltage sampling)
PAUSE_AT_END   = value
GAS_PRESSURE   = value (only used for PowerMax communications)
CUT_MODE       = value (only used for PowerMax communications)
```

It is possible to add new material, delete material, or edit existing material from the [PARAMETERS tab](#). It is also possible to achieve this by using [magic comments](#) in a g-Code file.

The material file may be edited with a text editor while LinuxCNC is running. After any changes have been saved, press **Reload** in the MATERIAL section of the [PARAMETERS Tab](#) to reload the material file.

5.7.9.7 Manual Material Handling

For manual material handling, the user would manually select the material from the materials list in the MATERIAL section of the [PARAMETERS Tab](#) before starting the G-code program. In addition to selecting materials with materials list in the MATERIAL section of the [PARAMETERS Tab](#), the user could use the MDI to change materials with the following command:

```
M190 Pn
```

The following code is the minimum code necessary to have a successful cut using the manual material selection method:

```
F#<_hal[plasmac.cut-feed-rate]>
M3 $0 S1
.
.
M5 $0
```

Note

Manual material handling will restrict the user to only one material for the entire job.

5.7.9.8 Automatic Material Handling

For automatic material handling, the user would add commands to their G-code file which will enable QtPlasmaC to change the material automatically.

The following codes may be used to allow QtPlasmaC to automatically change materials:

- **M190 Pn** - Changes the currently displayed material to material number n.
- **M66 P3 L3 Q1** - Adds a small delay (1 second in this example) to wait for QtPlasmaC to confirm that it successfully changed materials.
- **F#<_hal[plasmac.cut-feed-rate]>** - Sets the cut feed rate to the feed rate shown in the MATERIAL section of the [PARAMETERS Tab](#).

For automatic material handling, the codes **MUST** be applied in the order shown. If a G-code program is loaded which contains one or more material change commands then the first material will be displayed in the top header of the PREVIEW WINDOW on the [MAIN Tab](#) as the program is loading. The following code is the minimum code necessary to have a successful cut using the automatic material selection method:

```
M190 Pn
M66 P3 L3 Q1
F#<_hal[plasmac.cut-feed-rate]>
M3 $0 S1
.
.
M5 $0
```

5.7.9.9 Material Addition Via Magic Comments In G-code

By using "magic comments" in a G-code file it is possible to do the following:

- Add new materials to the <machine_name>_material.cfg file.
- Edit existing materials in the <machine_name>_material.cfg file.
- Use one or more temporary materials.

Temporary materials are numbered automatically by QtPlasmaC and the material change will also be done by QtPlasmaC and should not be added to the G-code file by CAM software or otherwise. The material numbers begin at 1000000 and are incremented for each temporary material. It is not possible to save a temporary material, however the user could create a new material while a temporary material is displayed and it will use the settings from the temporary material as the defaults.

Tip

It is possible to use temporary materials only and have an empty <machine_name>_material.cfg file. This negates the need to keep the QtPlasmaC materials file updated with the CAM tool file.

- The entire comment must be in parentheses.
- The beginning of the magic comment must be: **(o=**
- The equals sign must immediately follow each parameter with no space.
- The mandatory parameters must be in the magic comment (for option 0, **na** is optional and **nu** is not used).
- There can be any number and type of magic comments in a G-code file.
- If option 0 is to be used in addition to option 1 and/or option 2 then all option 0 must appear after all option 1 or all option 2 in the G-code file.

The options are:

Option	Description
0	Creates a temporary default material. Material information added with this option will be discarded by a LinuxCNC restart or materials reload. They may also be overwritten by a new G-code file that has temporary materials.
1	Adds a new material if the number specified does not exist.
2	Overwrites an existing material if the number specified exists. Adds a new material if the number specified does not exist.

Mandatory parameters are:

Name	Description
o	Selects the option to be used.
nu	Sets the material number (not used for option 0).
na	Sets the material name (optional for option 0).
ph	Sets the pierce height.
pd	Sets the pierce delay.
ch	Sets the cut height.
fr	Sets the feed rate.

Optional parameters are:

Name	Description
kw	Sets the kerf width.
th	Sets the THC status (0=disabled, 1=enabled).
ca	Sets the cut amps.
cv	Sets the cut voltage.
pe	Sets the pause at end delay.
gp	Sets the gas pressure (PowerMax).
cm	Sets the cut mode (PowerMax).
jh	Sets the puddle jump height.
jd	Sets the puddle jump delay.

A complete example:

```
(o=0, nu=2, na=5mm Mild Steel 40A, ph=3.1, pd=0.1, ch=0.75, fr=3000, kw=0.5, th=1, ca=45, ↵
cv=110, pe=0.1, gp=5, cm=1, jh=0, jd=0)
```

If a temporary material has been specified in a G-code file then the material change line (M190...) and wait for change line (M66...) will be added by the G-code filter and are not required in the G-code file.

5.7.9.10 Material Converter

This application is used to convert existing tool tables into QtPlasmaC material files. It can also create a material file from manual user input to entry fields.

At this stage the only conversions available are for tool tables exported from either SheetCam or Fusion 360.

SheetCam tool tables are complete and the conversion is fully automatic. The SheetCam tool file must be in the SheetCam .tools format.

Fusion 360 tool tables do not have all of the required fields so the user will be prompted for missing parameters. The Fusion 360 tool file must be in the Fusion 360 .json format.

If the user has a format from a different CAM software they would like converted, create a **New Topic** in the [PlasmaC forum](#) section of the [LinuxCNC forum](#) to request this addition.

Material Converter may be run from a terminal using one of the two following methods.

For a package installation (Buildbot) enter the following command in a terminal window:

```
qtplasmac-materials
```

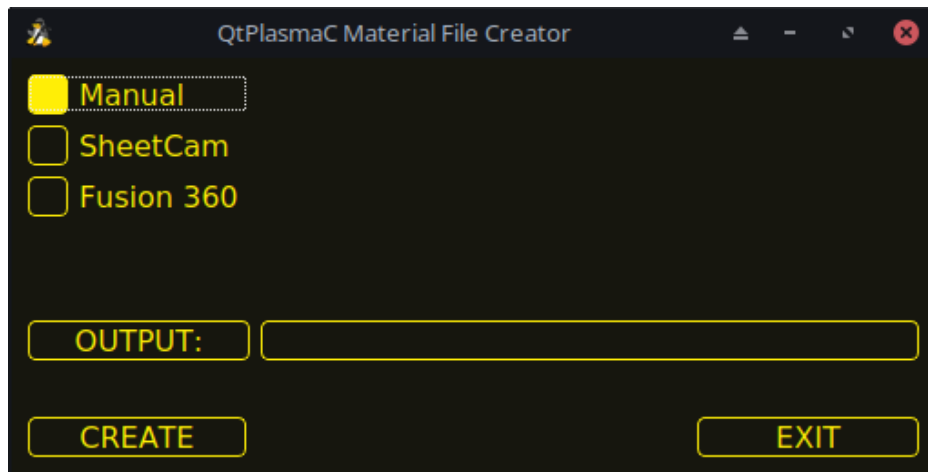
For a run in place installation enter the following two commands in a terminal window:

```
source ~/linuxcnc-dev/scripts/rip-environment
qtplasmac-materials
```

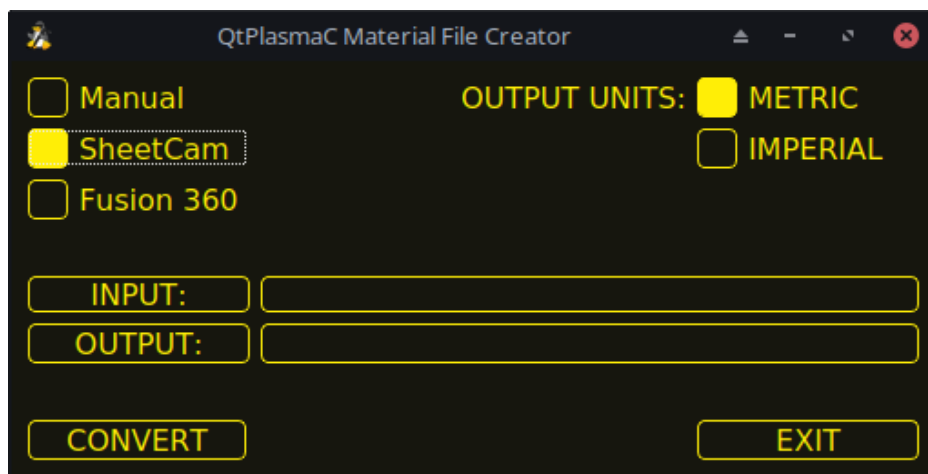
This will bring up the Material Converter Main dialog box with Manual selected as the default.

Select one of:

- **Manual** - to manually create a new material file.

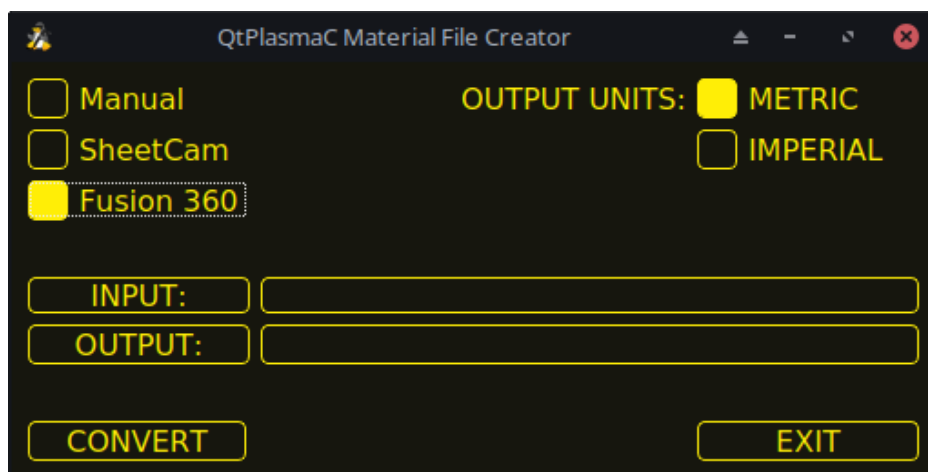


- **SheetCam** - to convert a SheetCam tool file.



For SheetCam only, select whether the user requires a metric or imperial output file.

- **Fusion 360** - to convert a Fusion 360 tool file.



To convert:

1. Select the Input File to be converted, press **INPUT** to bring up a file selector or directly enter the file in the entry box.

2. Select the Output File to write to, press **OUTPUT** to bring up a file selector or directly enter the file in the entry box. This would normally be ~/linuxcnc/configs/<machine_name>_material.cfg. If necessary, the user could select a different file and hand edit the <machine_name>_material.cfg file.
3. Click **CREATE/CONVERT** and the new material file will be created.

For both a Manual creation or a Fusion 360 conversion, a dialog box will show with all available parameters displayed for input. Any entry marked with *** is mandatory and all other entries are optional depending on the user's configuration needs.



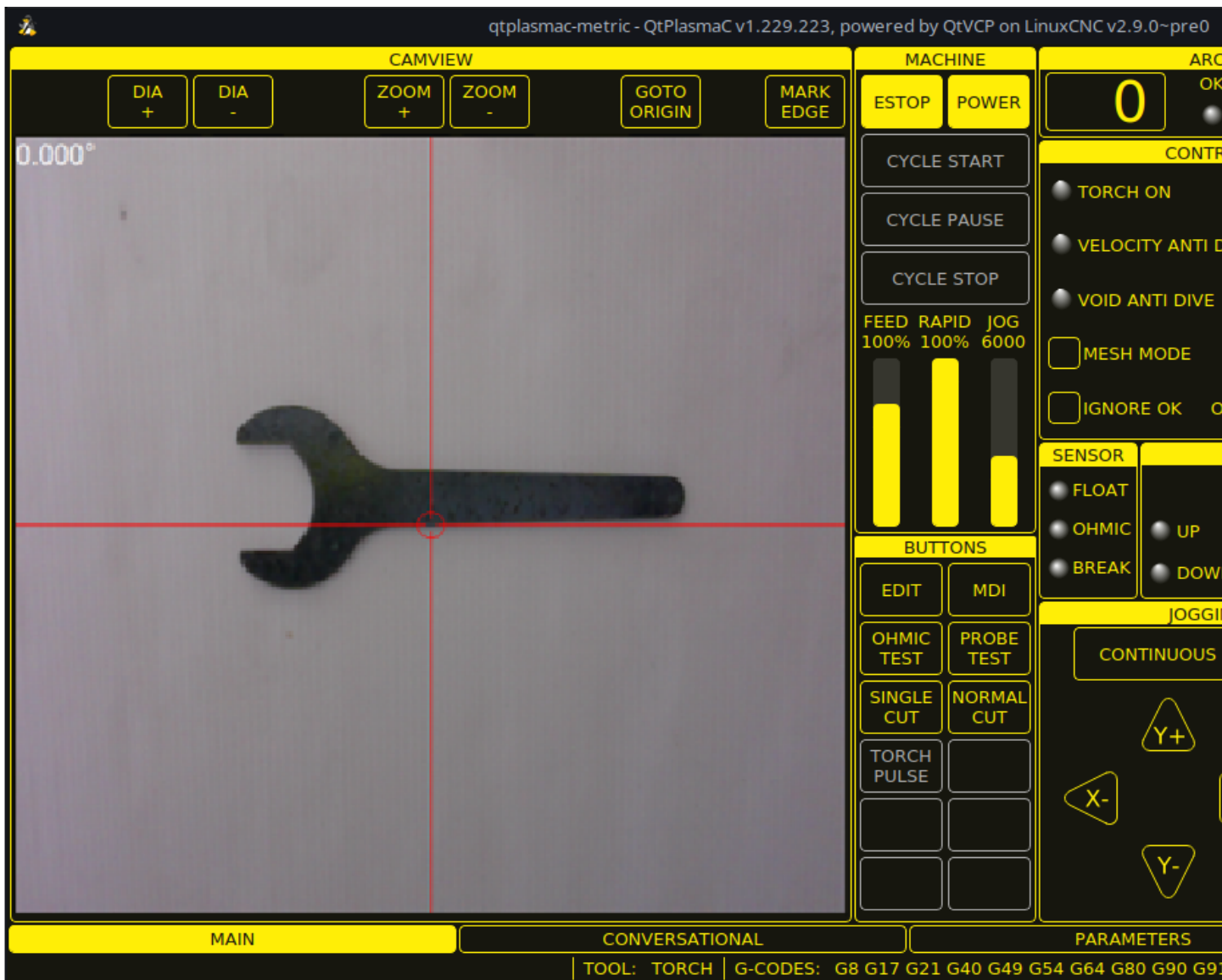
The image shows a 'Material Maker' dialog box with a dark background and yellow text. At the top, it says 'Items with a *** are mandatory'. Below this, there are several input fields, each with a label and a value. The labels are: 'Material Number ***' (value: 1), 'Material Name' (value: 0), 'Kerf Width' (value: 0), 'Pierce Height ***' (value: 0), 'Pierce Delay ***' (value: 0), 'Puddle Jump Height' (value: 0), 'Puddle Jump Delay' (value: 0), 'Cut Height ***' (value: 0), 'Cut Speed ***' (value: 0), 'Cut Amps' (value: 0), 'Cut Volts' (value: 0), 'Pause At End Of Cut' (value: 0), 'Gas Pressure' (value: 0), and 'Cut Mode' (value: 1). At the bottom, there are two buttons: 'Cancel' and 'OK'.

Parameter	Value
Material Number ***	1
Material Name	0
Kerf Width	0
Pierce Height ***	0
Pierce Delay ***	0
Puddle Jump Height	0
Puddle Jump Delay	0
Cut Height ***	0
Cut Speed ***	0
Cut Amps	0
Cut Volts	0
Pause At End Of Cut	0
Gas Pressure	0
Cut Mode	1

Buttons: Cancel, OK

Note

If the user selects `~/linuxcnc/configs/<machine_name>_material.cfg` and the file already exists, it will be overwritten.

5.7.9.11 CAMERA

QtPlasmaC has the ability to use a USB camera to set the origin with or without rotation compensation. The CAMERA button will be enabled after the machine is homed.

To use this feature, the user must set the camera's offset from the torch center by following the procedure described in [Peripheral Offsets](#).

To modify the offsets manually, the user could edit either or both the following lines in the [CAMERA_OFFSET] section of the `<machine_name>.prefs` file:

```
X axis = n.n
Y axis = n.n
```

where n.n is distance from the center line of the torch to the camera's cross hairs.

To set the origin with zero rotation:

1. Jog until the cross hairs are on top of the desired origin point.
2. Press **MARK EDGE**. The **MARK EDGE** button label will change to **SET ORIGIN** and the **GOTO ORIGIN** button will be disabled.
3. Press **SET ORIGIN**. The **SET ORIGIN** button label will change to **MARK EDGE** and the **GOTO ORIGIN** button will be enabled.
4. The torch will now move to the X0 Y0 position.
5. The offset is now successful set.

To set the origin with rotation:

1. Jog until the cross hairs are at the edge of the material a suitable distance away from the desired origin point.
2. Press **MARK EDGE**. The **MARK EDGE** button label will change to **SET ORIGIN** and the **GOTO ORIGIN** button will be disabled.
3. Jog until the cross hairs are at the origin point of the material.
4. Press **SET ORIGIN**. The **SET ORIGIN** button label will change to **MARK EDGE** and the **GOTO ORIGIN** button will be enabled.
5. The torch will now move to the X0 Y0 position.
6. The offset is now successfully set.

In the CAMVIEW panel, the mouse can affect the cross hairs and the zoom level as follows:

- Mouse Wheel Scroll - Change cross hair diameter.
- Mouse Wheel Button Double Click - Restores cross hair diameter to default.
- Mouse Left Button Clicked + Wheel Scroll - Changes camera zoom level.
- Mouse Left Button Clicked + Wheel Button Double Click - Restores default camera zoom level.

5.7.9.12 LASER

QtPlasmaC has the ability to use a laser to set the origin with or without rotation compensation. The LASER button will be enabled after the machine is homed.

To use this feature, the user must set the laser's offset from the torch center by following the procedure described in [Peripheral Offsets](#).

To modify the offsets manually, the user could edit either or both the following lines in the [LASER_OFFSET] section of the <machine_name>.prefs file:

```
X axis = n.n  
Y axis = n.n
```

where n.n is distance from the center line of the torch to the laser's cross hairs.

Additionally, the laser can be tied to any available output to turn the laser on and off via a HAL pin with the following name:

```
qtplasmac.laser_on
```

To set the origin with zero rotation:

1. Click the **LASER** button.
2. **LASER** button label will change to **MARK EDGE** and the HAL pin named `qtplasmac.laser_on` will be turned on.
3. Jog until the laser cross hairs are on top of the desired origin point.
4. Press **MARK EDGE**. The **MARK EDGE** button label will change to **SET ORIGIN**.
5. Press **SET ORIGIN**. The **SET ORIGIN** button label will change to **MARK EDGE** and the HAL pin named `qtplasmac.laser_on` will be turned off.
6. The torch will now move to the X0 Y0 position.
7. The offset is now successful set.

To set the origin with rotation:

1. Click the **LASER** button.
2. **LASER** button label will change to **MARK EDGE** and the HAL pin named `qtplasmac.laser_on` will be turned on.
3. Jog until the laser cross hairs are at the edge of the material a suitable distance away from the desired origin point.
4. Press **MARK EDGE**. The **MARK EDGE** button label will change to **SET ORIGIN**.
5. Jog until the laser cross hairs are at the origin point of the material.
6. Press **SET ORIGIN**. The **SET ORIGIN** button label will change to **MARK EDGE** and the HAL pin named `qtplasmac.laser_on` will be turned off.
7. The torch will now move to the X0 Y0 position.
8. The offset is now successfully set.

To turn the laser off and cancel an alignment:

1. Press the **LASER** button and hold for longer than 750 mSec.
2. **LASER** button label will change to **LASER** and the HAL pin named `qtplasmac.laser_on` will be turned off.
3. Release the **LASER** button.

If an alignment laser has been set up then it is possible to use the laser during [CUT RECOVERY](#) for accurate positioning of the new start coordinates.

5.7.9.13 Path Tolerance

Path tolerance is set with a G64 command and a following P value. The P value corresponds to the amount that the actual cut path followed by the machine may deviate from the programmed cut path. The default LinuxCNC path tolerance is set for maximum speed which will severely round corners when used with normal plasma cutting speeds.

It is recommended that the path tolerance is set by placing the appropriate G64 command and P value in the header of each G-code file.

The provided G-code filter program will test for the existence of a G64 Pn command prior to the first motion command. If no G64 command is found it will insert a G64 P0.1 command which sets the path tolerance to 0.1mm. For a imperial config the command will be G64 P0.004.

For Metric:

```
G64 P0.1
```

For Imperial:

```
G64 P0.004
```

5.7.9.14 Paused Motion

QtPlasmaC has the ability to allow the repositioning of the X and Y axes along the current cut path while the G-code program is paused.

In order to use this feature, LinuxCNC's Adaptive Feed Control (M52) must be turned on (P1).

To enable **Paused Motion** The preamble of the G-code must contain the following line:

```
M52 P1
```

To turn off **Paused Motion** at any point, use the following command:

```
M52 P0
```

5.7.9.15 Pause At End Of Cut

This feature can be used to allow the arc to "catch up" to the torch position to fully finish the cut. It is usually required for thicker materials and is especially useful when cutting stainless steel.

Using this feature will cause all motion to pause at the end of the cut while the torch is still on. After the dwell time (in seconds) set by the **Pause At End** parameter in the MATERIAL section of the [PARAMETERS Tab](#) has expired, QtPlasmaC will proceed with the M5 command to turn off and raise the torch.

5.7.9.16 Multiple Tools

QtPlasmaC has the ability to allow the use of more than one type of plasma tool by utilizing LinuxCNC spindles as a plasma tool when running a G-code program.

Valid plasma tools for use are:

Name	TOOL #	Description
Plasma Torch	0	Used for normal Plasma cutting.
Scribe	1	Used for material engraving.
Plasma Torch	2	Used for spotting (creating dimples to aid in drilling).

A LinuxCNC spindle number (designated by \$n) is required to be in the starting command and also the end command to be able to start and stop the correct plasma tool. Examples:

- **M3 \$0 S1** will select and start the plasma cutting tool.
- **M3 \$1 S1** will select and start the scribe.
- **M3 \$2 S1** will select and start the plasma spotting tool.
- **M5 \$0** will stop the plasma cutting tool.
- **M5 \$1** will stop the scribe.

- **M5 \$2** will stop the plasma spotting tool.

It is permissible to use **M5 \$-1** in lieu of the M5 \$n codes above to stop all tools.

In order to use a scribe, it is necessary for the user to add the X and Y axis offsets to the LinuxCNC tool table. Tool 0 is assigned to the Plasma Torch and Tool 1 is assigned to the scribe. Tools are selected with a **Tn M6** command, and then a **G43 H0** command is required to apply the offsets for the selected tool. It is important to note that the LinuxCNC tool table and tool commands only come into play if the user is using a [scribe](#) in addition to a plasma torch. For more information, see [scribe](#).

5.7.9.17 Velocity Reduction

There is a HAL pin available named **motion.analog-out-03** that can be changed in G-code with the **M67 (Synchronized with Motion)/M68 (Immediate)** commands. This pin will reduce the velocity to the percentage specified in the command.

It is important to thoroughly understand the difference between **Synchronized with Motion** and **Immediate**:

- **M67 (Synchronized with Motion)** - The actual change of the specified output (P2 (THC) for example) will happen at the beginning of the next motion command. If there is no subsequent motion command, the output changes will not occur. It is best practice to program a motion code (G0 or G1 for example) right after a M67.
- **M68 (Immediate)** - These commands happen immediately as they are received by the motion controller. Since these are not synchronized with motion, they will break blending. This means if these codes are used in the middle of active motion codes, the motion will pause to activate these commands.

Examples:

- **M67 E3 Q0** would set the velocity to 100% of **CutFeedRate**.
- **M67 E3 Q40** would set the velocity to 40% of **CutFeedRate**.
- **M67 E3 Q60** would set the velocity to 60% of **CutFeedRate**.
- **M67 E3 Q100** would set the velocity to 100% of **CutFeedRate**.

The minimum percentage allowed is 10%, values below this will be set to 10%.

The maximum percentage allowed is 100%, values above this will be set to 100%.

If the user intends to use this feature it would be prudent to add **M68 E3 Q0** to both the preamble and postamble of the G-code program so the machine starts and ends in a known state.



Important

G-CODE THC AND VELOCITY BASED THC ARE NOT ABLE TO BE USED IF CUTTER COMPENSATION IS IN EFFECT; AN ERROR MESSAGE WILL BE DISPLAYED.



Warning

If Cut Feed Rate in the MATERIAL section of the [PARAMETERS Tab](#) is set to Zero then QtPlasmaC will use **motion.requested-velocity** (as set by a standard Feedrate call in the G-code) for the THC calculations. This is not recommended as it is not a reliable way of implementing velocity based THC.

Note

All references to CutFeedRate refer to the **Cut Feed Rate** value displayed in the MATERIAL section of the [PARAMETERS Tab](#).

5.7.9.18 THC (Torch Height Controller)

The THC can be enabled or disabled from the THC frame of the [MAIN Tab](#).

The THC can also be enabled or disabled directly from the G-code program.

The THC does not become active until the velocity reaches 99.9% of the **CutFeedRate** and then the **THC Delay** time if any in the THC section of the [PARAMETERS Tab](#) has timed out. This is to allow the arc voltage to stabilize.

QtPlasmaC uses a control voltage which is dependent on the state of the **AUTO VOLTS** checkbox on the [MAIN Tab](#):

1. If **Use Auto Volts** is checked then the actual cut voltage is sampled at the end of the **THC Delay** time and this is used as the target voltage to adjust the height of the torch.
2. If **Use Auto Volts** is not checked then the voltage displayed as Cut Volts in the MATERIAL section of the [PARAMETERS Tab](#) is used as the target voltage to adjust the height of the torch.

G-code THC

THC may be disabled and enabled directly from G-code, provided the THC is not disabled in the THC Section of the [MAIN Tab](#), by setting or resetting the **motion.digital-out-02** pin with the M-Codes M62-M65:

- **M62 P2** will disable THC (Synchronized with Motion)
- **M63 P2** will enable THC (Synchronized with Motion)
- **M64 P2** will disable THC (Immediately)
- **M65 P2** will enable THC (Immediately)

It is important to thoroughly understand the difference between **Synchronized with Motion** and **Immediate**:

- **M62 and M63** (Synchronized with Motion) - The actual change of the specified output (P2 (THC) for example) will happen at the beginning of the next motion command. If there is no subsequent motion command, the output changes will not occur. It is best practice to program a motion code (G0 or G1 for example) right after a M62 or M63.
- **M64 and M65** (Immediate) - These commands happen immediately as they are received by the motion controller. Since these are not synchronized with motion, they will break blending. This means if these codes are used in the middle of active motion codes, the motion will pause to activate these commands.

Velocity Based THC

If the cut velocity falls below a percentage of **CutFeedRate** (as defined by the VAD Threshold % value in the THC frame of the CONFIGURATION section of the [PARAMETERS Tab](#)) the THC will be locked until the cut velocity returns to at least 99.9% of **CutFeedRate**. This will be made apparent by the **VELOCITY ANTI DIVE** indicator illuminating in the [CONTROL Panel](#) on the [MAIN Tab](#).

Velocity based THC prevents the torch height being changed when velocity is reduced for a sharp corner or a small hole.

It is important to note that [Velocity Reduction](#) affects the Velocity Based THC in the following ways:

1. If Velocity Reduction is invoked in the middle of the cut, the THC will be locked.
2. The THC will remain locked until the velocity reduction is canceled by returning it to a value that is above the **VAD Threshold**, and the torch actually reaches 99.9% of the **CutFeedRate**.

5.7.9.19 Cutter Compensation

LinuxCNC (QtPlasmaC) has the ability to automatically adjust the cut path of the current program by the amount specified in Kerf Width of the selected material's Cut Parameters. This is helpful if the G-code is programmed to the nominal cut path and the user will be running the program on different thickness materials to help ensure consistently sized parts.

To use cutter compensation the user will need to use G41.1, G42.1 and G40 with the kerf width HAL pin:

- **G41.1 D#<_hal[plasmac_run.kerf-width-f]>** : offsets torch to the left of the programmed path
- **G42.1 D#<_hal[plasmac_run.kerf-width-f]>** : offsets torch to the right of the programmed path
- **G40** turns the cutter compensation off



Important

IF **CUTTER COMPENSATION** IS IN EFFECT **G-CODE THC, VELOCITY BASED THC AND OVER CUT** ARE NOT ABLE TO BE USED; AN ERROR MESSAGE WILL BE DISPLAYED.

5.7.9.20 Initial Height Sense (IHS) Skip

Initial Height Sense may be skipped in one of two different ways:

1. If the THC is disabled, or the THC is enabled but not active, then the IHS skip will occur if the start of the cut is less than **Skip IHS** distance from the last successful probe.
2. If the THC is enabled and active, then the IHS skip will occur if the start of the cut is less than **Skip IHS** distance from the end of the last cut.

A value of zero for **Skip IHS** will disable all IHS skipping.

Any errors encountered during a cut will disable IHS skipping for the next cut if **Skip IHS** is enabled.

5.7.9.21 Probing

Probing may be done with either ohmic sensing or a float switch. It is also possible to combine the two methods, in which case the float switch will provide a fallback to ohmic probing. An alternative to ohmic probing is [Offset Probing](#)

If the machine's torch does not support ohmic probing, the user could have a separate probe next to the torch. In this case the user would extend the probe below the torch. The probe must NOT extend more than the minimum Cut Height below the torch and the Z axis offset distance needs to be entered as the **Ohmic Offset** in the PROBING frame of the CONFIGURATION section of the [PARAMETERS Tab](#).

Probing setup is done in the PROBING frame of the CONFIGURATION section of the [PARAMETERS Tab](#).

QtPlasmaC can probe at the full Z axis velocity so long as the machine has enough movement in the float switch to absorb any overrun. If the machine's float switch travel is suitable, the user could set the Probe Height to near the Z axis MINIMUM_LIMIT and do all probing at full speed.

Some float switches can exhibit a large switching hysteresis which shows up in the probing sequence as an excessive time to complete the final probe up.

- This time may be decreased by changing the speed of the final probe up.
- This speed defaults to 0.001mm (0.000039”) per servo cycle.
- It is possible to increase this speed by up to a factor of 10 by adding the following line to the custom.hal file:

```
setp plasmac.probe-final-speed n
```

where n is a value from 1-10. It is recommended to keep this value as low as possible.

Using this feature will change the final height slightly and will require thorough probe testing to confirm the final height.

This speed value affects ALL probing so if the user uses ohmic probing and the user changes this speed value then the user will need to probe test to set the require offset to compensate for this speed change as well as the float travel.

The reliability of this feature will only be as good as the repeatability of the float switch.

Note

Probe Height refers to the height above the Z axis MINIMUM_LIMIT.

5.7.9.22 Offset Probing

Offset Probing is the use of a probe that is offset from the torch. This method is an alternative to Ohmic Probing and uses the plasmac.ohmic-enable output pin to operate a solenoid for extending and retracting the probe. The plasmac.ohmic-probe input pin is used to detect the material and the **Ohmic Offset** in the PROBING frame of the CONFIGURATION section of the [PARAMETERS Tab](#) is used to set the correct measured height.

The probe could be a mechanically deployed probe, a permanently mounted proximity sensor or even simply a stiff piece of wire extending about 0.5mm (0.2”) below the torch tip. If the probe is mechanically deployed then it needs to extend/retract rather quickly to avoid excessive probing times and would commonly be pneumatically operated.

To use this feature, the user must set the probe’s offset from the torch center by following the procedure described in [Peripheral Offsets](#).

To modify the offsets manually, the user could edit either or both the following lines in the [OFFSET_PROBING] section of the <machine_name>.prefs file:

```
X axis = n.n
Y axis = n.n
Delay = t.t
```

where n.n is the offset of the probe from the torch center in machine units for the X and Y axes and t.t is the time in seconds to allow for any mechanical deployment of the probe if required.

Each of these parameters is optional and also may appear in any order. If a parameter is not detected then the default is 0.0. There can be no space after the X or Z, lower case is permissible.

When this variable appears in the INI file with either X or Y not equal to zero then QtPlasmaC will do **all** Ohmic Probing as Offset Probing. When a probe sequence has begun, the plasmac.ohmic-enable pin will be set True causing the probe to extend. When the material is detected the plasmac.ohmic-enable pin will be reset to false causing the probe to retract.

The probe will begin moving to the offset position simultaneously with the Z axis moving down to the Probe Height, probing will not commence unless the deployment timer has completed. It is required that the **Probe Height** in the PROBING frame of the CONFIGURATION section of the [PARAMETERS Tab](#) is above the top of the material to ensure that the probe is fully offset to the correct X/Y position before the final vertical probe down movement.

**Important**

PROBE HEIGHT NEEDS TO BE SET ABOVE THE TOP OF THE MATERIAL FOR OFFSET PROBING

5.7.9.23 Cut Types

QtPlasmaC allows two different cut modes:

1. **NORMAL CUT** - runs the loaded G-code program to pierce then cut.
2. **PIERCE ONLY** - only pierces the material at each cut start position, useful prior to a **NORMAL CUT** on [thick materials](#)

There are two ways of enabling this feature:

1. Utilize the default [custom user button](#) to toggle between the cut types.
2. Adding the following line to the G-code program before the first cut to enable **Pierce Only** mode for the current file:

```
#<pierce-only> = 1
```

If using a custom user button is utilized then QtPlasmaC will automatically reload the file when the cut type is toggled.

5.7.9.24 Hole Cutting - Intro

It is recommended that any holes to be cut have a diameter no less than one and a half times the thickness of the material to be cut.

It is also recommended that holes with a diameter of less than 32mm (1.26") are cut at 60% of the feed rate used for profile cuts. This should also lock out THC due to velocity constraints.

QtPlasmaC can utilize G-code commands usually set by a CAM Post Processor (PP) to aid in hole cutting or if the user does not have a PP or the user's PP does not support these methods then QtPlasmaC can automatically adapt the G-code to suit. This automatic mode is disabled by default.

There are three methods available for improving the quality of small holes:

1. **Velocity Reduction** - [Reducing the velocity](#) to approximately 60% of the **CutFeedRate**.
2. **Arc Dwell (Pause At End)** - Keeping the torch on for a short time at the end of the hole while motion is stopped to allow the arc to catch up.
3. **Over cut** - Turning the torch off at the end of the hole then continue along the path.

Note

If both **Arc Dwell** and **Over cut** are active at the same time then **Over cut** will take precedence.

**Important**

OVER CUT IS NOT ABLE TO BE USED IF CUTTER COMPENSATION IS IN EFFECT; AN ERROR MESSAGE WILL BE DISPLAYED.

5.7.9.25 Hole Cutting

G-code commands can be set up by either by a CAM Post Processor (PP) or by hand coding.

Hole Cutting Velocity Reduction

If cutting a hole requires a reduced velocity then the user would use the following command to set the velocity: **M67 E3 Qnn** where nn is the percentage of the velocity desired. For example, **M67 E3 Q60** would set the velocity to 60% of the current material's **CutFeedRate**.

See the [Velocity Based THC](#) section.

Sample code:

```
G21 (metric)
G64 P0.005
M52 P1 (allow paused motion)
F#<_hal[plasmac.cut-feed-rate]> (feed rate from cut parameters)
G0 X10 Y10
M3 $0 S1 (start cut)
G1 X0
M67 E3 Q60 (reduce feed rate to 60%)
G3 I10 (the hole)
M67 E3 Q0 (restore feed rate to 100%)
M5 $0 (end cut)
G0 X0 Y0
M2 (end job)
```

Arc Dwell (Pause At End)

This method can be invoked by setting the [Pause At End](#) parameter in the MATERIAL frame of the [PARAMETERS Tab](#).

Over cut

The torch can be turned off at the end of the hole by setting the **motion.digital-out-03** pin with the M-Codes **M62 (Synchronized with Motion)** or **M64 (Immediate)**. After turning the torch off it is necessary to allow the torch to be turned on again before beginning the next cut by resetting the **motion.digital-out-03** pin with the M-Codes **M63** or **M65**, this will be done automatically by the QtPlasmaC G-code parser if it reaches an M5 command without seeing a **M63 P3** or **M65 P3**.

After the torch is turned off the hole path will be followed for a default length of 4mm (0.157"). This distance may be specified by adding **#<oclength> = n** to the G-code file.

- **M62 P3** will turn the torch off (Synchronized with Motion)
- **M63 P3** will allow the torch to be turned on (Synchronized with Motion)
- **M64 P3** will turn the torch off (Immediately)
- **M65 P3** will allow the torch to be turned on (Immediately)

It is important to thoroughly understand the difference between **Synchronized with motion** and **Immediate**:

- **M62 and M63** (Synchronized with Motion) - The actual change of the specified output (P2 (THC) for example) will happen at the beginning of the next motion command. If there is no subsequent motion command, the output changes will not occur. It is best practice to program a motion code (G0 or G1 for example) right after a M62 or M63.
- **M64 and M65** (Immediate) - These commands happen immediately as they are received by the motion controller. Since these are not synchronized with motion, they will break blending. This means if these codes are used in the middle of active motion codes, the motion will pause to activate these commands.

Sample code:

```
G21 (metric)
G64 P0.005
M52 P1 (allow paused motion)
F#<_hal[plasmac.cut-feed-rate]> (feed rate from cut parameters)
G0 X10 Y10
M3 $0 S1 (start cut)
G1 X0
M67 E3 Q60 (reduce feed rate to 60%)
G3 I10 (the hole)
M62 P3 (turn torch off)
G3 X0.8 Y6.081 I10 (continue motion for 4mm)
M63 P3 (allow torch to be turned on)
M67 E3 Q0 (restore feed rate to 100%)
M5 $0 (end cut)
G0 X0 Y0
M2 (end job)
```

5.7.9.26 Hole Cutting - Automatic

QtPlasmaC has the ability to automatically modify the G-code to reduce the velocity and/or apply **Over cut** which can be useful when cutting holes.

For valid hole sensing it is required that all values in the G2 or G3 G-code line are explicit, an error dialog will be displayed if any values are mathematically calculated.

QtPlasmaC Hole Sensing is disabled by default. It can be enabled/disabled by using the following G-code parameters to select the desired hole sensing mode:

- **#<holes> = 0** - Causes QtPlasmaC to disable hole sensing if it was previously enabled.
- **#<holes> = 1** - Causes QtPlasmaC to reduce the speed of holes less than 32mm (1.26") to 60% of **CutFeedRate**.
- **#<holes> = 2** - Causes QtPlasmaC to **Over cut** the hole in addition to the velocity changes in setting 1.
- **#<holes> = 3** - Causes QtPlasmaC to reduce the speed of holes less than 32mm (1.26") and arcs less than 16mm (0.63") to 60% of **CutFeedRate**.
- **#<holes> = 4** - Causes QtPlasmaC to **Over cut** the hole in addition to the velocity change in setting 3.

The default hole size for QtPlasmaC hole sensing is 32mm (1.26"). It is possible to change this value with the following command in a G-code file:

- **#<h_diameter> = nn** - To set a diameter (nn) in the same units system as the rest of the G-code file.

The default velocity for QtPlasmaC small holes is 60% of the current feed rate. It is possible to change this value with the following command in a G-code file:

- **#<h_velocity> = nn** - to set the percentage (nn) of the current feed rate required.

Over cut

If Hole Sensing modes 2 or 4 are active, QtPlasmaC will over cut the hole in addition to the velocity changes associated with modes 1 and 3.

The default over cut length for QtPlasmaC hole sensing is 4mm (0.157"). It is possible to change this value with the following command in a G-code file:

- **#<oclength> = nn** to specify an over cut length (nn) in the same units system as the rest of the G-code file.

Arc Dwell (Pause At End)

This feature can be used in addition to setting the desired hole sensing mode via the appropriate G-code parameter by setting the [Pause At End](#) parameter in the MATERIAL frame of the [PARAMETERS Tab](#).

Sample code:

```
G21 (metric)
G64 P0.005
M52 P1 (allow paused motion)
F#<_hal[plasmac.cut-feed-rate]> (feed rate from cut parameters)
#<holes> = 2 (over cut for holes)
#<oclength> = 6.5 (optional, 6.5mm over cut length)
G0 X10 Y10
M3 $0 S1 (start cut)
G1 X0
G3 I10 (the hole)
M5 $0 (end cut)
G0 X0 Y0
M2 (end job)
```

Note

It is OK to have multiple and mixed hole commands in a single G-code file.

5.7.9.27 Single Cut

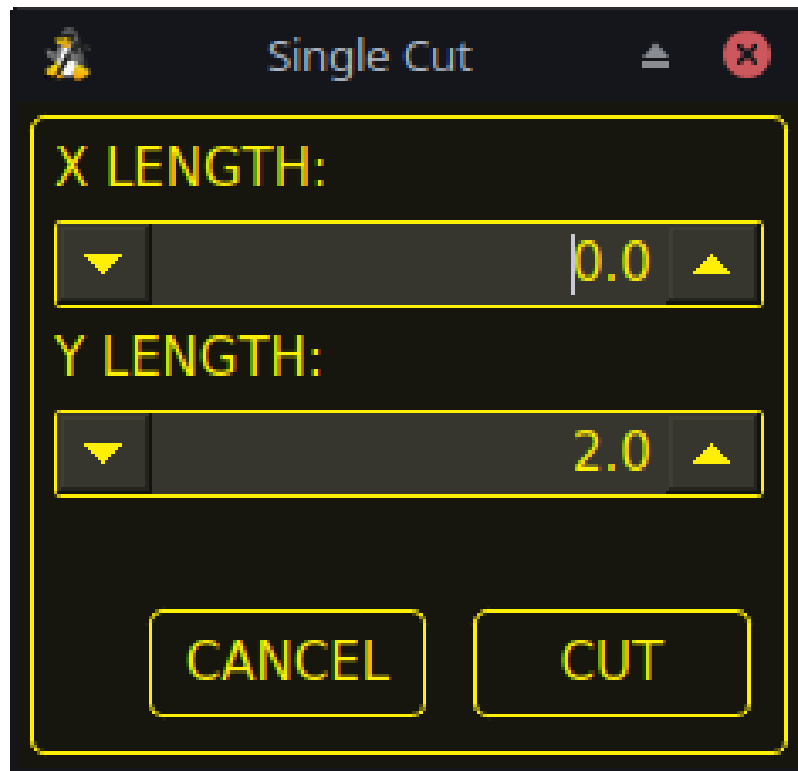
A single cut is a single unidirectional cutting move often used to cut a sheet into smaller pieces prior to running a G-code program.

The machine needs to be homed before commencing a single cut.

A single cut will commence from the machine's current X/Y position.

Automatic Single Cut

This is the preferred method. The parameters for this method are entered in the following dialog box that is displayed after pressing a [user button](#) which has been coded to run single cut:



1. Jog to the required X/Y start position.
2. Set required appropriate material, or edit the Feed Rate for the default material in the [PARAM-ETERS Tab](#).
3. Press the assigned single cut user button.
4. Enter the length of the cut along the X and/or Y axes.
5. Press the **CUT** button and the cut will commence.

Pendant Single Cut

If the machine is equipped with a pendant that can start and stop the spindle plus jog the X and Y axes, the user can manually perform a single cut.

1. Jog to the required X/Y start position.
2. Set the required feed rate with the Jog Speed slider.
3. Start the cut process by starting the spindle.
4. After probing the torch will fire.
5. When the Arc OK is received the machine can be jogged along the cut line using the jog buttons.

6. When the cut is complete stop the spindle.
7. The torch will turn off and the Z axis will return to the starting position.

Manual Single Cut

Manual single cut requires that either [keyboard shortcuts](#) are enabled in the GUI SETTINGS section of the [PARAMETERS Tab](#), or a custom user button is specified as a [manual cut](#) button.

If the user is using a custom user button then substitute **F9** with **User Button** in the following description.

1. Jog to the required X/Y start position.
2. Start the procedure by pressing **F9**. The jog speed will be automatically set to the feed rate of the currently selected material. The jog label will blink to indicate that the jog speed is temporarily being overridden (jog speed manipulation will be disabled while a manual cut is active). **CYCLE START** will change to **MANUAL CUT** and blink.
3. After probing the torch will fire.
4. When the Arc OK is received the machine can be jogged along the cut line using the jog keys.
5. The Z height will remain locked at the cut height for the duration of the manual cut, regardless of the Torch Height Controller **ENABLE** status.
6. When the cut is complete press **F9** or **Esc** or the **CYCLE STOP** button.
7. The torch will turn off and the Z axis will return to the starting position.
8. The jog speed will automatically be returned to the value it was prior to initiating the manual cut process, the label will stop blinking and the jog speed manipulation will be enabled. **MANUAL CUT** will stop blinking and revert to **CYCLE START**.

Note

If the torch flames out during cutting, the user must still press **F9** or **Esc** or the **CYCLE STOP** button to end the cut. This clears the Z offsets and returns the torch to the starting position.

5.7.9.28 Thick Materials

Cutting thick materials can be problematic in that the large amount of molten metal caused by piercing can shorten the life of consumables and also may cause a puddle high enough that the torch may hit the puddle while moving to cut height.

There are two functions built into QtPlasmaC to help alleviate these issues.

Pierce Only

Pierce Only mode converts the loaded G-code program and then runs the program to pierce the material at the start position of each cut. Scribe and Spotting commands will be ignored and no pierce will take place in those locations.

This mode is useful for thick materials which may produce enough dross on the material surface from piercing to interfere with the torch while cutting. The entire sheet can be pierced and then cleaned off prior to cutting.

It is possible to use near-end-of-life consumables for piercing and then they can be swapped out for good consumables to be used while cutting.

Pierce Only is one of two different [cut types](#)

Puddle Jump

Puddle Jump is the height that the torch will move to after piercing and prior to moving to **Cut Height** and is expressed as a percentage of **Pierce Height**. This allows the torch to clear any puddle of molten material that may be caused by piercing. The maximum allowable height is 200% of the **Pierce Height**

Setting for **Puddle Jump** are described in [cut parameters](#)

The recommended option is to use **Pierce Only** due to it being able to utilise near end of life consumables.



Important

PUDDLE JUMP IS DISABLED DURING CUT RECOVERY

5.7.9.29 Mesh Mode (Expanded Metal Cutting)

QtPlasmaC is capable of cutting of expand (mesh) metal provided the machine has a pilot arc torch and it is capable of Constant Pilot Arc (CPA) mode.

Mesh Mode disables the THC and also ignores a lost Arc OK signal during a cut. It can be selected by checking the **Mesh Mode** check button in the CONTROL section of the [MAIN Tab](#).

If the machine has [RS485](#) communications enabled with a Hypertherm PowerMax plasma cutter, selecting **Mesh Mode** will automatically override the **Cut Mode** for the currently selected material and set it to cut mode 2 (CPA). When **Mesh Mode** is disabled, the **Cut Mode** will be return to the default cut mode for the currently selected material.

It is also possible to start a **Mesh Mode** cut without receiving an Arc OK signal by checking the **Ignore Arc OK** check button in the CONTROL section of the [MAIN Tab](#).

Both **Mesh Mode** and **Ignore Arc OK** can be enabled/disabled at any time during a job.

5.7.9.30 Ignore Arc OK

Ignore Arc OK mode disables the THC, will begin a cut without requiring an Arc OK signal, and will ignore a lost Arc OK signal during a cut.

This mode can be selected by:

1. Checking the **Ignore Arc OK** check button in the CONTROL section of the [MAIN Tab](#).
2. Setting HAL pin **motion.digital-out-01** to 1 via G-code.
 - **M62 P1** will enable **Ignore Arc OK** (Synchronized with Motion)
 - **M63 P1** will disable **Ignore Arc OK** (Synchronized with Motion)
 - **M64 P1** will enable **Ignore Arc OK** (Immediately)
 - **M65 P1** will disable **Ignore Arc OK** (Immediately)

It is important to thoroughly understand the difference between **Synchronized with motion** and **Immediate**:

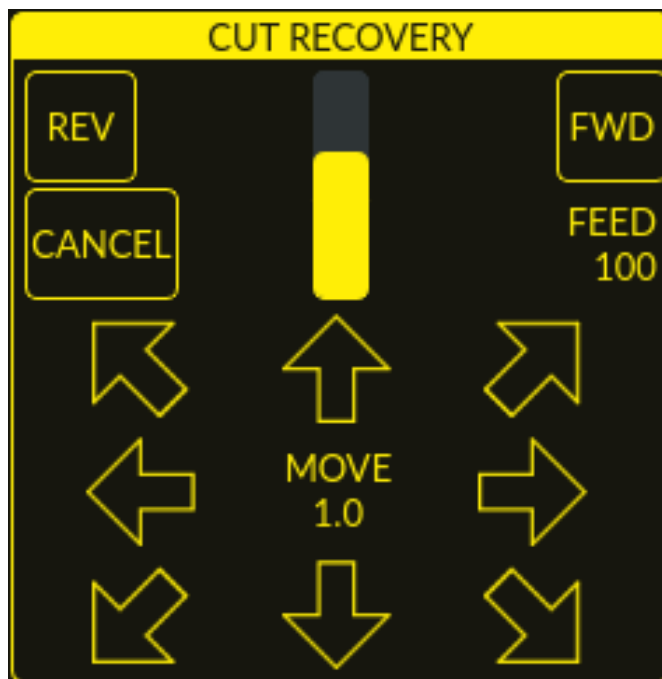
- **M62 and M63** (Synchronized with Motion) - The actual change of the specified output (P2 (THC) for example) will happen at the beginning of the next motion command. If there is no subsequent motion command, the output changes will not occur. It is best practice to program a motion code (G0 or G1 for example) right after a M62 or M63.
-

- **M64 and M65** (Immediate) - These commands happen immediately as they are received by the motion controller. Since these are not synchronized with motion, they will break blending. This means if these codes are used in the middle of active motion codes, the motion will pause to activate these commands.

This mode may also be used in conjunction with **Mesh Mode** if the user doesn't require an Arc OK signal to begin the cut.

Both **Mesh Mode** and **Ignore Arc OK** can be enabled/disabled at any time during a job.

5.7.9.31 Cut Recovery



This feature will produce a CUT RECOVERY panel that will allow the torch to be moved away from the cut path during a [paused motion](#) event in order to position the torch over a scrap portion of the material being cut so that the cut restarts with a minimized arc-divot. The CUT RECOVERY panel will display automatically over top of the JOGGING panel when motion is paused.

It is preferable to make torch position adjustments from the point at which paused motion occurred, however if moving along the cut path is necessary prior to setting the new start point, the user may use the paused motion controls (**REV**, **FWD**, and a **JOG-SPEED** slider) at the top of the CUT RECOVERY panel. Once the user is satisfied with the positioning of the torch along the cut path, moving off the cut path is achieved by pressing the **DIRECTION** buttons. Each press of the **DIRECTION** button will move the torch a distance equivalent to the **Kerf Width** parameter of the currently selected material.

The moment the torch has been moved off the cut path, the paused motion controls (**REV**, **FWD**, and a **JOG-SPEED** slider) at the top of the CUT RECOVERY panel will become disabled.

Once the torch position is satisfactory, press **CYCLE RESUME** and the cut will resume from the new position and travel the shortest distance to the original paused motion location. The CUT RECOVERY panel will close and the JOGGING panel will display when the torch returns to the original paused motion location.

Pressing **CANCEL MOVE** will cause the torch to move back to where it was positioned before the direction keys were used to offset the torch. It will not reset any **REV** or **FWD** motion.

Pressing **CYCLE STOP** will cause the torch to move back to where it was positioned before the direction keys were used to offset the torch and the CUT RECOVERY panel overlay will return to the JOGGING panel. It will not reset any **REV** or **FWD** motion.

If an alignment laser has been set up then it is possible to use the laser during cut recovery for very accurate positioning of the new start coordinates. If either the X axis offset or Y axis offset for the laser would cause the machine to move out of bounds then an error message will be displayed.

To use a laser for cut recovery when paused during a cut:

1. Click the **LASER** button.
2. **LASER** button will change to disabled, the HAL pin named qtplasmac.laser_on will be turned on and the X and Y axis will offset so that the laser cross hairs will indicate the starting coordinates of the cut when it is resumed.
3. Continue the cut recovery as described above.

If a laser offset is in effect when **CANCEL MOVE** is pressed then this offset will also be cleared.

Note

Cut recovery movements will be limited to a radius of 10mm (0.4") from either the point the program was paused, or from the last point on the cut path if paused motion was used.



Important

PUDDLE JUMP IS DISABLED DURING CUT RECOVERY

5.7.9.32 Run From Line

If the user has the Run From Line option enabled in the GUI SETTINGS section of the [PARAMETERS Tab](#) they will have the ability to start from any line in a G-code program via the following methods:

1. Clicking any line in the Preview Window
2. Clicking any line in the G-code Window

Note that the Run From Line function will run from the beginning of the selected line.

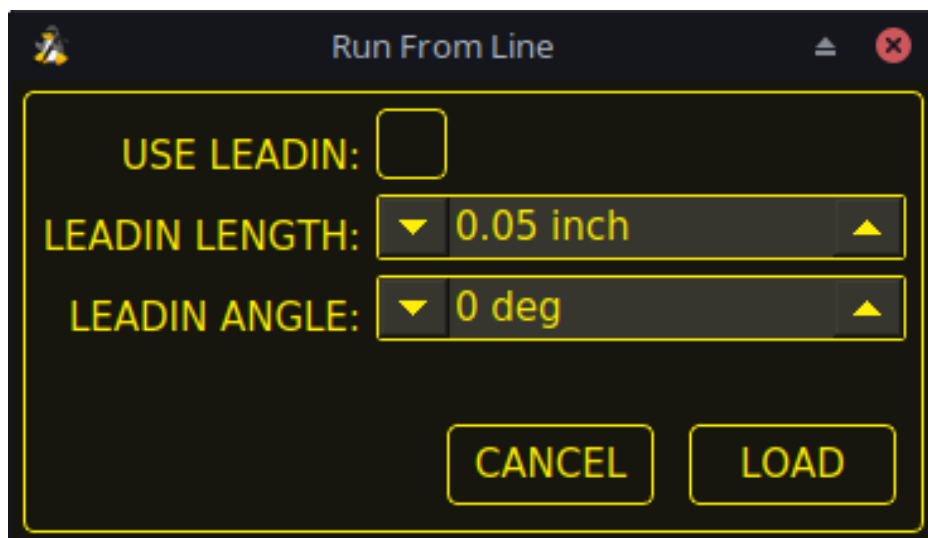
It is important to note that G-code programs can be run from any selected line using this method, however a leadin may not be possible depending on the line selected. In this case, an error message will be displayed to let the user know the leadin calculation was not possible.

Once the user has selected the starting place, the **CYCLE START** button will blink "**SELECTED nn**" where nn is the corresponding line number selected. Clicking this button will bring up the following Run From Line dialog box:

It is not possible to use Run From Line from within a subroutine. If the user selects a line within a subroutine and clicks "**SELECTED nn**" then an error message will be displayed that includes the O-Code name of the subroutine.

It is not possible to use Run From Line if previous G-code has set cutter compensation active. If the user selects a line while cutter compensation is active and clicks "**SELECTED nn**" then an error message will be displayed.

It is possible to select a new line while Run From Line is active.



Name	Description
USE LEADIN	This radio button will allow the user to start the selected line with a leadin.
LEADIN LENGTH	If USE LEADIN is selected, this will set the length of the lead in the machine units.
LEADIN ANGLE	If USE LEADIN is selected, this will set the angle of approach for the leadin. The angle is measured such that positive increases in value move the leadin counter-clockwise: 0 Degrees = 3 o'clock position 90 Degrees = 12 o'clock position 180 Degrees = 9 o'clock position 270 Degrees = 6 o'clock position
CANCEL	This button will cancel the Run From Line dialog box and any selections.
LOAD	This button will load a temporary "rfl.ngc" program with any selected leadin parameters applied. If the leadin cannot be calculated for the selected line, the following error message will be displayed: "Unable to calculate a leadin for this cut Program will run from selected line with no leadin applied"

After pressing **LOAD**, the blinking "SELECTED nn" button will change to **RUN FROM LINE CYCLE START** button. Click this button to start the program from the beginning of the selected line.

Run From Line selections may be canceled in the following ways:

1. Click the background of the preview window - this method will cancel a selection of either a cut line in the preview window, or a G-code line in the G-code window.
2. Click the text of the first line of the G-code program in the G-code display - this method will cancel a selection of either a cut line in the preview window, or a G-code line in the G-code window.
3. Clicking **RELOAD** in the G-code window header - this method will cancel the Run From Line process if LOAD was clicked on the Run From Line dialog box and "rfl.ngc" is displayed as the loaded file name in the G-code window header. This will return the user to the originally loaded file.

5.7.9.33 Scribe

A scribe may be operated by QtPlasmaC in addition to the plasma torch.

Using a scribe requires the use of the LinuxCNC tool table. Tool 0 is assigned to the plasma torch and Tool 1 is assigned to the scribe. The scribe X and Y axes offsets from the plasma torch need to be entered into the LinuxCNC tool table. This is done by editing the tool table via the main GUI, or by editing the **tool.tbl** file in the **<machine_name>** configuration directory. This will be done after the scribe can move to the work piece to help determine the appropriate offset.

The plasma torch offsets for X and Y will always be zero. The tools are selected by the **Tn M6** command followed by a **G43 H0** command which is required to apply the offsets. The tool is then started with a **M3 \$n S1** command. For **n**, use 0 for torch cutting or 1 for scribing.

To stop the scribe, use the G-code command **M5 \$1**.

If the user has not yet assigned the HAL pins for the scribe in the configuration wizard then they may do so by using the appropriate [configuration wizard](#) or by manually editing the HAL file, see [modifying QtPlasmaC](#).

There are two HAL output pins used to operate the scribe, the first pin is used to arm the scribe which moves the scribe to the surface of the material. After the [Arm Delay](#) has elapsed, the second pin is used to start the scribe. After the [On Delay](#) has elapsed, motion will begin.

Using QtPlasmaC after enabling the scribe requires the selection of either the torch or the scribe in each G-code file as a LinuxCNC tool.

The first step is to set the offsets for the scribe by following the procedure described in [Peripheral Offsets](#).

The final step is to set the [scribe delays](#) required:

1. **Arm Delay** - allows time for the scribe to descend to the surface of the material.
2. **On Delay** - allows time for the scribe to start before motion begins.

Save the parameters in the Config tab.

After the above directions are completed, the scribe may be tested manually by issuing a **M3 \$1 S1** command in the MDI input. The user may find it helpful to use this method to scribe a small divot and then try to pulse the torch in the same location to align the offsets between the scribe and the torch.

To use the scribe from G-code:

```
...
M52 P1 (paused motion on)
F#<_hal[plasmac.cut-feed-rate]>
T1 M6 (select scribe)
G43 H0 (apply offsets for current tool)
M3 $1 S1 (start the scribe)
.
M5 $1 (stop the scribe)
.
T0 M6 (select torch)
G43 H0 (apply offsets for current tool)
G0 X0 Y0 (parking position)
M5 $-1 (end all)
```

It is a good idea to switch back to the torch at the end of the program before the final rapid parking move so the machine is always in the same state at idle.

The user can switch between the torch and the scribe any number of times during a program by using the appropriate G-codes.

Issuing **M3 S1** (without \$n) will cause the machine to behave as if an **M3 \$0 S1** had been issued and issuing **M5** (without \$n) will cause the machine to behave as if an **M5 \$0** had been issued. This will control the torch firing by default in order to provide backward compatibility for previous G-code files.

**Warning**

If there is an existing manual tool change parameter set in the <machine_name>.hal file then QtPlasmaC will convert it to an automatic tool change.

5.7.9.34 Spotting

To achieve spotting to mark the material prior to drilling etc., QtPlasmaC can pulse the torch for a short duration to mark the spot to drill.

Spotting can be configured by following these steps:

1. Set the arc voltage **Threshold** in the Spotting section of the [PARAMETERS Tab](#). Setting the voltage threshold to zero will cause the delay timer to begin immediately upon starting the torch. Setting the voltage threshold above zero will cause the delay timer to begin when the arc voltage reaches the threshold voltage.
2. Set the **Time On** in the Spotting section of the [PARAMETERS Tab](#). When the **Time On** timer has elapsed, the torch will turn off. Times are adjustable from 0 to 9999 milliseconds.

The torch is then turned on in G-code with the **M3 \$2 S1** command which selects the plasma torch as a spotting tool.

To turn the torch off, use the G-code command **M5 \$2**.

For more information on multiple tools, see [multiple tools](#).

LinuxCNC (QtPlasmaC) requires some motion between any **M3** and **M5** commands. For this reason, a minimal movement at a high speed is required to be programmed.

An example G-code is:

```
G21 (metric)
F99999 (high feed rate)
.
.
G0 X10 Y10
M3 $2 S1 (spotting on)
G91 (relative distance mode)
G1 X0.000001
G90 (absolute distance mode)
M5 $2 (spotting off)
.
.
G0 X0 Y0
G90
M2
```

Note

The **high feed rate** of 99999 is to ensure that the motion is at the machine's highest feed rate.

**Important**

SOME PLASMA CUTTERS WILL NOT BE SUITABLE FOR THIS FEATURE.
IT IS RECOMMENDED THAT THE USER CARRY OUT SOME TEST SPOTTING TO ENSURE THAT THE PLASMA CUTTER IS CAPABLE OF UTILIZING THIS FEATURE.

5.7.9.35 Virtual Keyboard Custom Layouts

Virtual keyboard support is available for only the "onboard" onscreen keyboard. If it is not already on the system it may be installed by typing the following in a terminal:

```
sudo apt install onboard
```

The following two custom layouts are used for soft key support:



Figure 5.33: Number keypad - used for the CONVERSATIONAL Tab and the PARAMETERS Tab

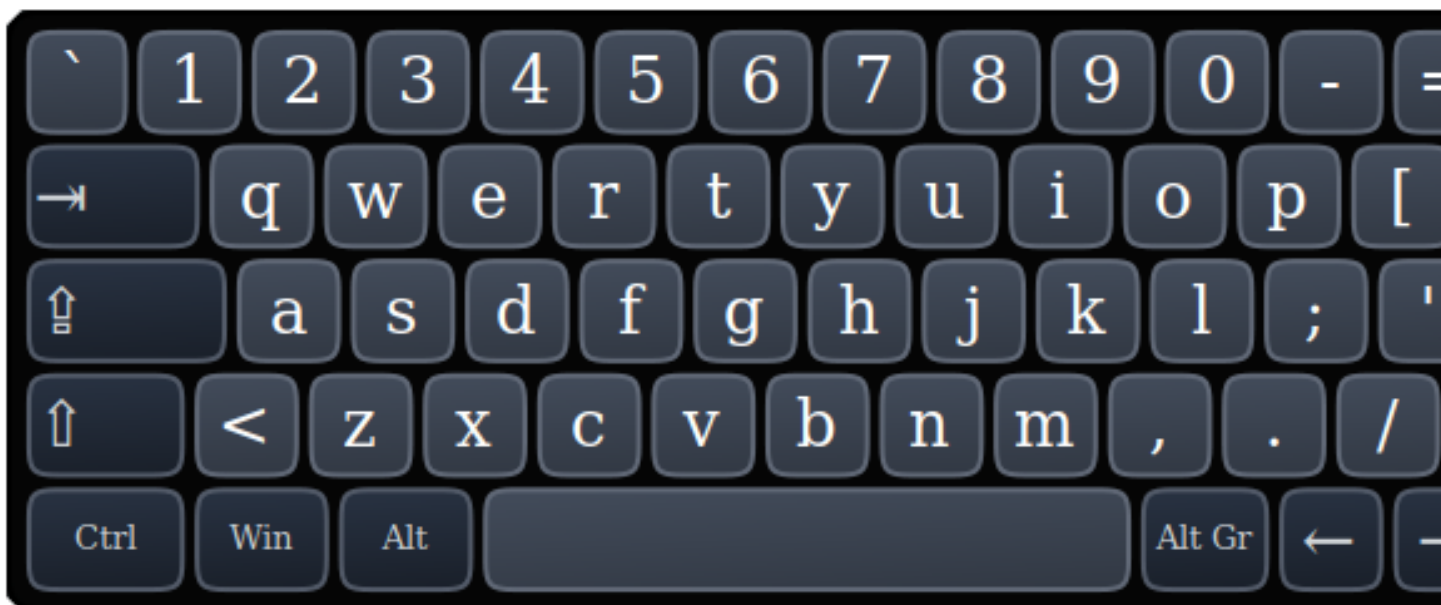


Figure 5.34: Alpha-numeric keypad - used for G-code editing and file management.

If the virtual keyboard has been repositioned and on the next opening of a virtual keyboard it is not visible then clicking twice on the onboard icon in the system tray will reposition the virtual keyboard so the move handle is visible.

5.7.9.36 Keyboard Shortcuts

Below is a list of all available keyboard shortcuts in QtPlasmaC.

Note

All keyboard shortcuts are disabled by default.

In order to utilize them, **KB Shortcuts** must be enabled in the **GUI SETTINGS** section of the [PARAMETERS Tab](#).

Keyboard Shortcut	Action
ESC	Aborts current automated motion (example: a running program, a probe test, etc.) as well as an active torch pulse (behaves the same as clicking CYCLE STOP).
F1	Toggles the GUI E-STOP button (if the GUI E-STOP button is enabled).
F2	Toggles the GUI power button.
F9	Toggles the "Cutting" command, used to begin or end a manual cut.
F12	Show stylesheet editor.
ALT + RETURN	Places QtPlasmaC into Manual Data Input (MDI) mode. Note that ALT + ENTER will achieve the same result. In addition, pressing RETURN (or ENTER) with no entry in the MDI will close the MDI window.
1	Changes jog speed to 10% of the value present in the DEFAULT_LINEAR_VELOCITY variable in the [DISPLAY] section of the <machine_name>.ini file.
2	Changes jog speed to 20% of the value present in the DEFAULT_LINEAR_VELOCITY variable in the [DISPLAY] section of the <machine_name>.ini file.
3	Changes jog speed to 30% of the value present in the DEFAULT_LINEAR_VELOCITY variable in the [DISPLAY] section of the <machine_name>.ini file.
4	Changes jog speed to 40% of the value present in the DEFAULT_LINEAR_VELOCITY variable in the [DISPLAY] section of the <machine_name>.ini file.
5	Changes jog speed to 50% of the value present in the DEFAULT_LINEAR_VELOCITY variable in the [DISPLAY] section of the <machine_name>.ini file.
6	Changes jog speed to 60% of the value present in the DEFAULT_LINEAR_VELOCITY variable in the [DISPLAY] section of the <machine_name>.ini file.
7	Changes jog speed to 70% of the value present in the DEFAULT_LINEAR_VELOCITY variable in the [DISPLAY] section of the <machine_name>.ini file.
8	Changes jog speed to 80% of the value present in the DEFAULT_LINEAR_VELOCITY variable in the [DISPLAY] section of the <machine_name>.ini file.
9	Changes jog speed to 90% of the value present in the DEFAULT_LINEAR_VELOCITY variable in the [DISPLAY] section of the <machine_name>.ini file.

Keyboard Shortcut	Action
0	Changes jog speed to 100% of the value present in the DEFAULT_LINEAR_VELOCITY variable in the [DISPLAY] section of the <machine_name>.ini file.
SHIFT+1	Changes rapid speed to 10%.
SHIFT+2	Changes rapid speed to 20%.
SHIFT+3	Changes rapid speed to 30%.
SHIFT+4	Changes rapid speed to 40%.
SHIFT+5	Changes rapid speed to 50%.
SHIFT+6	Changes rapid speed to 60%.
SHIFT+7	Changes rapid speed to 70%.
SHIFT+8	Changes rapid speed to 80%.
SHIFT+9	Changes rapid speed to 90%.
SHIFT+0	Changes rapid speed to 100%.
CTRL+1	Changes feed rate to 10%.
CTRL+2	Changes feed rate to 20%.
CTRL+3	Changes feed rate to 30%.
CTRL+4	Changes feed rate to 40%.
CTRL+5	Changes feed rate to 50%.
CTRL+6	Changes feed rate to 60%.
CTRL+7	Changes feed rate to 70%.
CTRL+8	Changes feed rate to 80%.
CTRL+9	Changes feed rate to 90%.
CTRL+0	Changes feed rate to 100%.
CTRL+H or HOME	Homes all axes if they are not yet homed and have a homing sequence set in the <machine_name>.ini file. If they are already homed, they will no longer be homed.
CTRL+R	Cycle Start if the program is not already running. Cycle Resume if the program is paused.
END	Touches off X and Y to 0.
DEL	Allows the user to use a laser to set an origin with or without rotation. See the LASER section for detailed instructions.
SPACE BAR	Pause motion.
O	Opens a new program.
L	Loads the previously opened program if no program is loaded. Reloads the current program if there is a program loaded.
→	Jogs the X axis positive.
←	Jogs the X axis negative.
↑	Jogs the Y axis positive.
↓	Jogs the Y axis negative.
PAGE UP	Jogs the Z axis positive.
PAGE DOWN	Jogs the Z axis negative.
[Jogs the A axis positive.
]	Jogs the A axis negative.
.	Jogs the B axis positive.
,	Jogs the B axis negative.
SHIFT (+ Jog Key)	The shift key is used with any jog key to invoke a rapid jog.
+ (+ Jog Key)	The plus key can be used with any jog key to invoke a rapid jog (behaves the same as SHIFT).
- (+ Jog Key)	The minus key can be used with any jog key to invoke a slow jog (10% of the displayed jog speed) If SLOW jogging is already active, the axis will jog at the displayed jog speed.

5.7.9.37 MDI

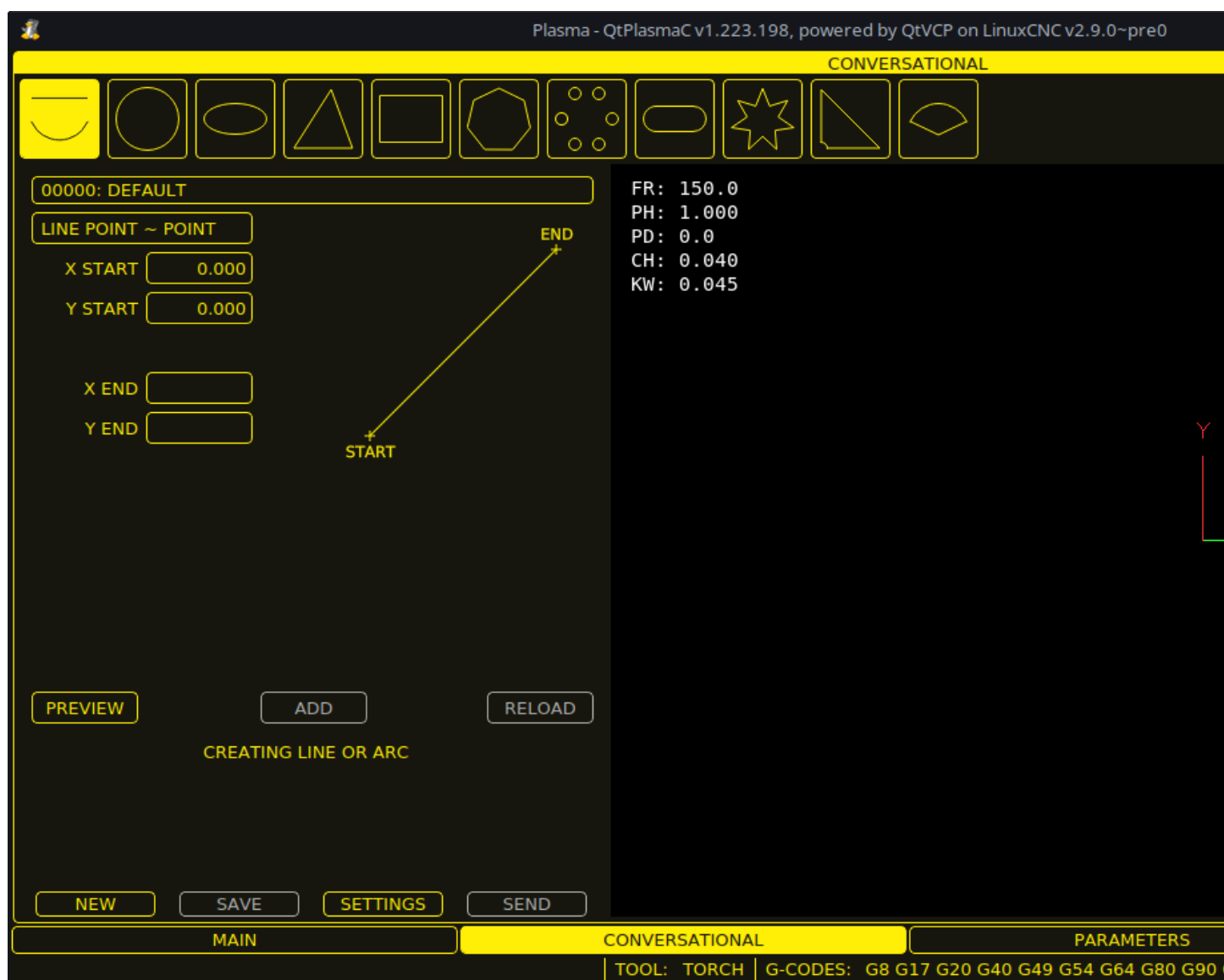
In addition to the typical G and M codes that are allowed by LinuxCNC in MDI mode, the MDI in QtPlasmaC can be used to access several other handy features. The following link outlines the features and their use: [MDI Line Widget](#)

Note

M3, M4, and M5 are not allowed in the QtPlasmaC MDI.

In addition, pressing RETURN (or ENTER) with no entry in the MDI will close the MDI window.

5.7.10 Conversational Shape Library



The **Conversational Shape Library** consists of several basic shapes and functions to assist the user with generating quick G-code at the machine to cut simple shapes quickly. This feature is found on the [CONVERSATIONAL Tab](#).

Note

The Conversational Library is not meant to be a CAD/CAM replacement as there are limitations to what can be achieved.

Blank entries in the shape input boxes will use the current setting at the time the G-code was generated. For example, if **X start** was left blank then the current X axis position would be used.

All leadins and leadouts are arcs except for **Circles** and **Stars**:

Circles:

- If the circle is external then any leadin or leadout will be an arc.
- If the circle is internal and a **small hole** then any leadin will be perpendicular and there will be no lead out.
- If the circle is internal and not a **small hole** then any leadin and leadout will be an arc. If the leadin has a length greater than half the radius then the leadin will revert to perpendicular and there will be no leadout. If the leadout has a length greater than half the radius then there will be no leadout.

Stars:

- The leadin is at the same angle as the first cut and the leadout is at the same angle as the last cut.

Note

A **small hole** is a circle that is smaller than the SMALL HOLE DIAMETER specified in the CONVERSATIONAL SETTINGS page.

Note

The holes in a BOLT CIRCLE shape will also abide by the above rules.

The cut order will occur in the same order as the shape was built.

Pressing **Return** on the keyboard while editing parameters will automatically show the preview of the shape if there are enough parameters entered to create the shape. Clicking any of the available check boxes will do the same.

The general functions are as follows:

Name	Description
Material Drop Down	Allows the user to select the desired material for cutting. If "VIEW MATERIAL" is selected on the PARAMETERS Tab , a visual reference showing key material cut settings will be displayed on the Conversational Preview Window. Examples are: Feed Rate, Pierce Height, Pierce Delay, Cut Height, and Kerf Width (for Conversational only). Cut Amps will be shown if PowerMax communications are enabled.
NEW	Removes the current G-code file and load a blank G-code file.
SAVE	Opens a dialog box allowing the current shape to be saved as a G-code file.
SETTINGS	Allows the changing of the global settings.
SEND	Loads the current shape into LinuxCNC (QtPlasmaC). If the last edit was not added then it will be discarded.
PREVIEW	Displays a preview of the current shape provided the required information is present.

Name	Description
CONTINUE	This button is used for lines and arcs only. Allows another segment to be added to the current segment/segments.
ADD	Stores the current shape into the current job.
UNDO	Reverts to the previously stored state.
RELOAD	Reloads the original G-code file or a blank file if none was loaded.

If there is a G-code file loaded in LinuxCNC (QtPlasmaC) when the [CONVERSATIONAL Tab](#) is selected, that code will be imported into the conversational as the first shape of the job. If this code is not required then it can be removed by pressing the **NEW** button.

If there is an added shape that is unsaved or unsent then it is not possible to switch tabs in the GUI. To re-enable switching tabs it is necessary to either **SAVE** the shape, **SEND** the shape, or press **NEW** to remove the shape.

If **NEW** is pressed to remove an added shape that is unsaved or unsent then a warning dialog will be displayed.

Note

All distances are in machine units relative to the current User Coordinate System and all angles are in degrees.

5.7.10.1 Conversational Settings

Global settings for the shape library can be set by pressing the **SETTINGS** button in the [CONVERSATIONAL Tab](#). This will display all of the available settings parameters that are used for G-code program creation. These include:

- **Preamble**
- **Postamble**
- **Origin (Center or Bottom Left)**
- **Leadin length**
- **Leadout length**
- **Small hole diameter**
- **Small hole speed**
- **Preview Window Grid Size**

Any internal circle that has a diameter less than **Small hole diameter** is classified as a small hole and will have a straight leadin with a length that is the lesser of either the radius of the hole or the specified leadin length. It will also have its feed rate set to **Small hole speed**.

Preamble and Postamble may be entered as a string of G-Codes and M-Codes separate by spaces. If the user wishes for the generated G-code to have each code on an individual line then this is made possible by separating the codes with **\n**.

This will place all codes on the same line:

```
G21 G40 G49 G64p0.1 G80 G90 G92.1 G94 G97
```

This will place each code on its own line:

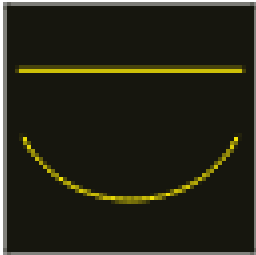
```
G21\nG40\nG49\nG64p0.1\nG80\nG90\nG92.1\nG94\nG97
```

Pressing the **RELOAD** button will discard any changed but unsaved settings.

Pressing the **SAVE** button will save all the settings as displayed.

Pressing the **EXIT** button will close the setting panel and return to the previous shape.

5.7.10.2 Conversational Lines And Arcs



Lines and arcs have an additional option in that they may be strung together to create a complex shape.

There are two line types and three arc types available:

1. **Line** given a start point and an end point.
2. **Line** given a start point, length, and angle.
3. **Arc** given a start point, way point, and end point.
4. **Arc** given a start point, end point, and radius.
5. **Arc** given a start point, length, angle, and radius.

To use lines and arcs:

1. Select the **Lines and Arcs** icon.
2. Select the type of line or arc to create.
3. Choose the material from the MATERIAL drop down. If no material is chosen, the default material (00000) will be used.
4. Enter the desired parameters.
5. Press **PREVIEW** to see the shape.
6. If satisfied with the shape press **CONTINUE**.
7. Change the line or arc type if needed and continue this procedure until the shape is complete.
8. Press **SEND** to send the G-code file to LinuxCNC (QtPlasmaC) for cutting.

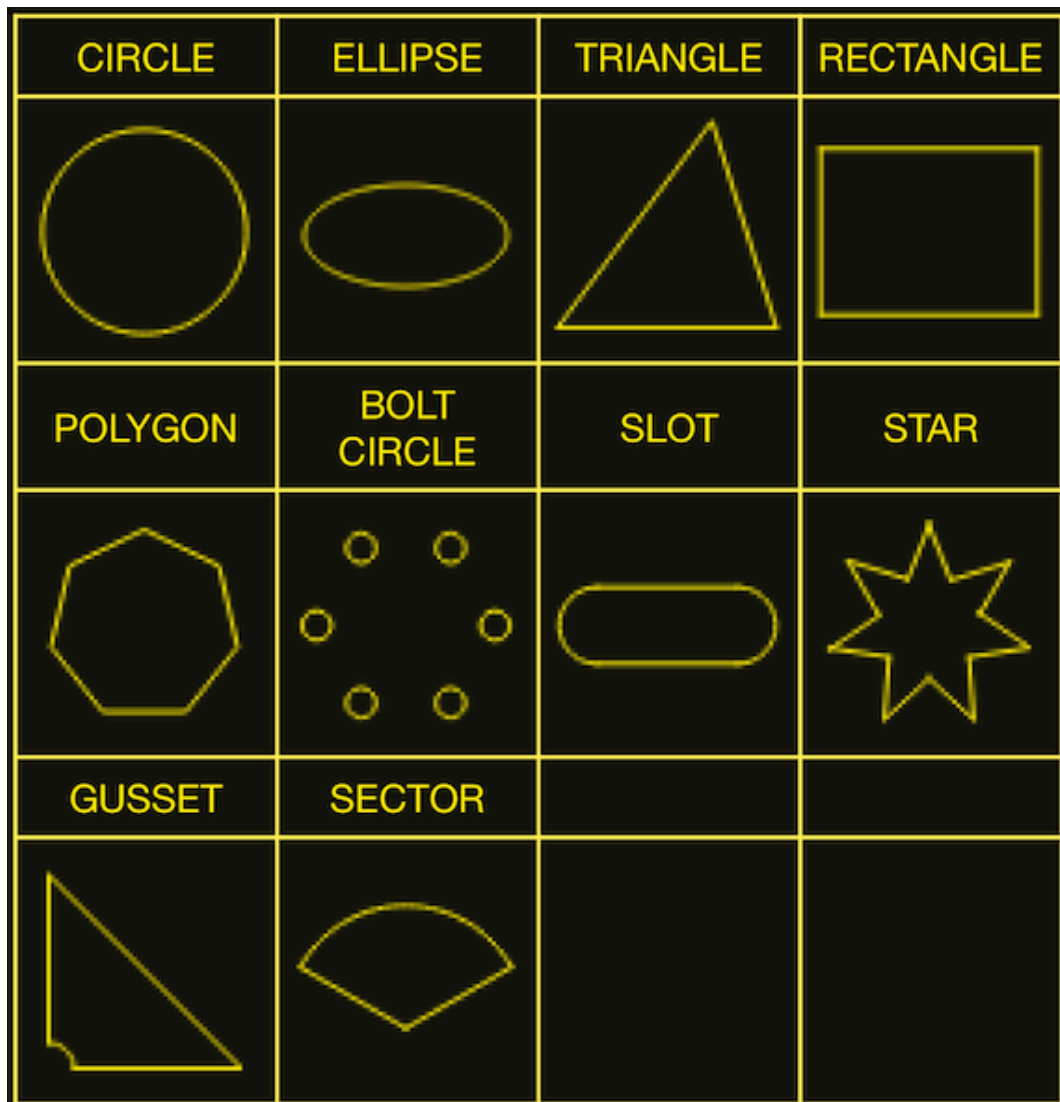
If the user wishes to create a closed shape, they will need to create any required leadin as the first segment of the shape. If a leadout is required it will need to be the last segment of the shape.

Note

At this stage there is no automatic option for a leadin/leadout creation if the shape is closed.

5.7.10.3 Conversational Single Shape

The following shapes are available for creation:



To create a shape:

1. Select the corresponding icon for the shape to create. The available parameters will be displayed.
2. Choose the material from the MATERIAL drop down. If no material is chosen, the default material (00000) will be used.
3. Enter the appropriate values and press **PREVIEW** to display the shape.
4. If the shape is not correct, edit the values and press **PREVIEW** and the new shape will be displayed. Repeat until satisfied with the shape.
5. Press **ADD** to add the shape to the G-code file.
6. Press **SEND** to send the G-code file to LinuxCNC (QtPlasmaC) for cutting.

For **CIRCLE**, the **OVER CUT** button will become valid when a CUT TYPE of INTERNAL is selected and the value entered in the DIAMETER field is less than the Small Hole Diameter parameter in the Conversational SETTINGS section.

For **BOLT CIRCLE** the **OVER CUT** button will become valid if the value entered in the HOLE DIA field is less than the SMALL HOLES DIAMETER parameter in the Conversational SETTINGS section.

For the following shapes, KERF OFFSET will become active once a LEAD IN is specified:

1. TRIANGLE
2. RECTANGLE
3. POLYGON
4. SLOT
5. STAR
6. GUSSET

5.7.10.4 Conversational Group Of Shapes

Multiple shapes can be added together to create a complex group.

The cut order of the group is determined by the order in which the individual shapes are added to the group.

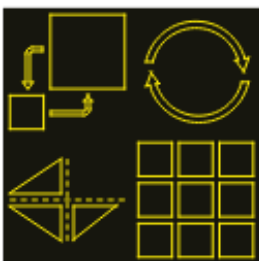
Once a shape is added to the group it cannot be edited or removed.

Groups cannot have shapes removed, only added to.

To create a group of shapes:

1. Create the first shape as in **Single Shape**.
2. Press **ADD** and the shape will be added to the group.
3. If the user wishes to add another version of the same shape then edit the required parameters and press **ADD** when satisfied with the shape.
4. If the user wishes to add a different shape, select that shape and create it as in **Single Shape**.
5. Repeat until all the required shapes to complete the group have been added.
6. Press **SEND** to send the G-code file to LinuxCNC (QtPlasmaC) for cutting.

5.7.10.5 Conversational Block



The Conversational Block feature allows block operations to be performed on the current shape or group of shapes displayed in the [CONVERSATIONAL Tab](#). This can include a G-code file not created using the Conversational Shape Library that has been previously loaded from the [MAIN Tab](#).

A previously saved Block G-code file may also be loaded from the [MAIN Tab](#) and then have any of its operations edited using the Conversational Block feature.

Block operations:

- Rotate
- Scale
- Array
- Mirror
- Flip

To create a block:

1. Create a shape, a group, or use a previously loaded G-code file.
2. Click the **Block** icon to open the Block tab.
3. Enter the appropriate values in the Block tab and press **PREVIEW** to display the resulting changes.
4. If the result is not correct, edit the values and press **PREVIEW** and the new result will be shown. Repeat until satisfied with the result.
5. Press **ADD** to complete the procedure.
6. Press **SEND** to send the G-code file to LinuxCNC (QtPlasmaC) for cutting, or **SAVE** to save the G-code file.

COLUMNS & ROWS - specifies the number of duplicates of the original shape arranged in columns and rows as well as the offset distance from the original shape.

ORIGIN - offset the result from the origin coordinates.

ANGLE - rotate the result.

SCALE - scale the result.

ROTATION - rotate the shape within the result.

MIRROR - mirror the shape about its X coordinates within the result.

FLIP - flip the shape about its Y coordinates within the result.

If the result is an array of shapes then the cut order of the result is from the left column to the right column, starting at the bottom row and ending at the top row.

5.7.10.6 Conversational Saving A Job

The current job displayed in the Preview Panel may be saved at any time by using the bottom **SAVE** button. If the G-code has been sent to LinuxCNC (QtPlasmaC) and the user has left the [CONVERSATIONAL Tab](#), the user may still save the G-code file from the GUI. Alternatively, the user could click the [CONVERSATIONAL Tab](#) which will reload the job, at which time they can press the **SAVE** button.

5.7.11 Error Messages

5.7.11.1 Error Logging

All errors are logged into the machine log which is able to be viewed in the [STATISTICS Tab](#). The log file is saved into the configuration directory when QtPlasmaC is shutdown. The five last logfiles are kept, after which the oldest logfile is deleted each time a new log file is created. These saved log files may be viewed with any text editor.

5.7.11.2 Error Message Display

By default, QtPlasmaC will display error messages via a Operator Error popup window. In addition, QtPlasmaC will alert the user that an error has been sent to the machine log by displaying the message **"ERROR SENT TO MACHINE LOG"** in the lower left portion of the status bar.

The user may opt to disable the Operator Error popup window, and view the error messages by going to the [STATISTICS Tab](#) by changing the following preference to **False** in the **[SCREEN_OPTIONS]** of the <machine_name>.prefs file in the <machine_name> directory:

```
desktop_notify
```

Note

<machine_name>.prefs must be edited with QtPlasmaC closed or any changes will be overwritten on exit.

Additionally, it is possible for **ERROR SENT TO MACHINE LOG** to flash to get the user's attention by adding or editing the following line in the [GUI_OPTIONS] section of the <machine_name>.prefs file:

```
Flash error = True
```

5.7.11.3 Critical Errors

There are a number of error messages printed by QtPlasmaC to inform the user of faults as they occur. The messages can be split into two groups, **Critical** and **Warning**.

Critical Errors will cause the running program to pause, and the operator will need to clear the cause of the error before proceeding.

If the error was received during cutting then forward or reverse motion is allowed while the machine is paused to enable the user to reposition the machine prior to resuming the cut.

When the error is cleared the program may be resumed.

These errors indicate the corresponding sensor was activated during cutting:

- **breakaway switch activated program is paused**
- **float switch activated program is paused**
- **ohmic probe activated program is paused**

These errors indicate the corresponding sensor was activated before probing commenced:

- **ohmic probe detected before probing program is paused**
- **float switch detected before probing program is paused**
- **breakaway switch detected before probing program is paused**

The Arc OK signal was lost during cutting motion, before the **M5** command was reached:

- **valid arc lost program is paused**

The Z axis reached the bottom limit before the work piece was detected:

- **bottom limit reached while probing down program is paused**

The work piece is too high for any safe rapid removes:

- **material too high for safe traverse program is paused**

One of these values in MATERIAL section of the [PARAMETERS Tab](#) is invalid (For example: if they are set to zero):

- **invalid pierce height or invalid cut height or invalid cut volts program is paused**

No arc has been detected after attempting to start the number of times indicated by **Max Starts** in the ARC frame of the CONFIGURATION section of the [PARAMETERS Tab](#):

- **no arc detected after <n>d start attempts program is paused**
- **no arc detected after <n>d start attempts manual cut is stopped**

THC has caused the bottom limit to be reached while cutting:

- **bottom limit reached while THC moving down program is paused**

THC has caused the top limit to be reached while cutting:

- **top limit reached while THC moving up program is paused**

These errors indicate move to pierce height would exceed the Z Axis MAX_LIMIT for the corresponding probe method:

- **pierce height would exceed Z axis maximum limit condition found while moving to probe height during float switch probing**
- **pierce height would exceed Z axis maximum limit condition found while moving to probe height during ohmic probing**

These errors indicate the move to pierce height would exceed the Z axis maximum safe height for the corresponding probe method:

- **pierce height would exceed Z axis maximum safe height condition found while float switch probing**
- **pierce height would exceed Z axis maximum safe height condition found while ohmic probing**

5.7.11.4 Warning Errors

Warning errors will not pause a running program and are informational only.

These errors indicate the corresponding sensor was activated before a probe test commenced:

- **ohmic probe detected before probing probe test aborted**
- **float switch detected before probing probe test aborted**
- **breakaway switch detected before probing probe test aborted**

This indicates that the corresponding sensor was activated during a consumable change:

- **breakaway, float, or ohmic activated during consumable change, motion is paused**
WARNING: MOTION WILL RESUME IMMEDIATELY UPON RESOLVING THIS CONDITION!

**Warning**

CONSUMABLE CHANGE MOTION WILL RESUME IMMEDIATELY UPON RESOLVING THE CORRESPONDING SENSOR ACTIVATION.

This indicates that the corresponding sensor was activated during probe testing:

- **breakaway switch detected during probe test**

This indicates that probe contact was lost before probing up to find the zero point:

- **probe trip error while probing**

This indicates that the bottom limit was reached during a probe test:

- **bottom limit reached while probe testing**

This indicates that the move to pierce height would exceed the Z Axis MAX_LIMIT during the corresponding probe method:

- **pierce height would exceed Z axis maximum limit condition found while moving to probe height during float switch probe testing**
- **pierce height would exceed Z axis maximum limit condition found while moving to probe height during ohmic probe testing**

This indicates that the safe height has been reduced due to THC raising the Z axis during cutting:

- **safe traverse height has been reduced**

This indicates that the value for the Arc Voltage was invalid (NAN or INF) when QtPlasmaC launched.

- **invalid arc-voltage-in**

5.7.12 Updating QtPlasmaC

5.7.12.1 Standard Update

QtPlasmaC update notices are posted at: <https://forum.linuxcnc.org/plasmac/37233-plasmac-updates>

Users are strongly encouraged to create a Username and subscribe to the above thread to receive update notices.

For a standard ISO installation, LinuxCNC will only be updated when a new minor release has been released. QtPlasmaC will then automatically update its configuration the first time it is run after a LinuxCNC update.

LinuxCNC is normally updated by entering the following commands into a terminal window (one at a time):

```
sudo apt update
sudo apt dist-upgrade
```

5.7.12.2 Continuous Update

Enhancements and bug fixes will not be available on a standard installation until a new minor release of LinuxCNC has been released. If the user wishes to update whenever a new QtPlasmaC version has been pushed, they could use the LinuxCNC Buildbot repository rather than the standard LinuxCNC repository by following the instructions at: <http://buildbot.linuxcnc.org/>

5.7.13 Modify An Existing QtPlasmaC Configuration

There are two ways to modify an existing QtPlasmaC configuration:

1. Running the appropriate [configuration wizard](#) and loading the conf file saved by the wizard.
2. Manually edit the .ini and/or the .hal file of the configuration.



Important

Any manual modification to the <machine_name>.ini and <machine_name>.hal files will not be registered in PnCConf or StepConf.

Note

If unsure of the HAL pin's full name, the user may start LinuxCNC and run **HalShow** for a full listing of all HAL pins.

5.7.14 Customizing QtPlasmaC GUI

Styling of the QtPlasmaC GUI is done with Qt stylesheets and some customization may be achieved by the use of a custom stylesheet. This allows the user to change some GUI items such as color, border, size, etc. It cannot change the layout of the GUI.

Information on Qt stylesheets is available [here](#).

There are two methods available to apply custom styles:

1. Add A Custom Style: use this for minor style changes.
2. Create A New Style use this for a complete style change.

5.7.14.1 Add A Custom Style

Adding style changes to the default stylesheet is achieved by creating a file in the <machine_name> configuration directory. This file MUST be named qtplasmac_custom.qss. Any required style changes are then added to this file.

For example the user may want the arc voltage display in red, a green Torch On LED of a larger size and a larger Torch Enable button. This would be done with the following code in qtplasmac_custom.qss:

```
#arc_voltage {
    color: #ff0000 }

#led_torch_on {
    qproperty-diameter: 30;
    qproperty-color: green }

#torch_enable::indicator {
    width: 30;
    height: 30}
```

5.7.14.2 Create A New Style

Custom stylesheets are enable by a setting in the [GUI_OPTIONS] section of the <machine_name>.prefs file giving the filename of the stylesheet.

Custom style = the_cool_style.qss

The filename may be any valid filename. The standard extension name is .qss but this is not mandatory. There are some constraints on the custom stylesheet for QtPlasmaC, e.g., the jog buttons, cut-recovery buttons, and the conversational shape buttons are image files and are not able to be custom styled.

The custom style file requires a header in the following format:

```

/*****
Custom Stylesheet Header

color1 = #000000
#QtPlasmaC default = #ffee06

color2 = #e0e0e0
#QtPlasmaC default = #16160e

color3 = #c0c0c0
#QtPlasmaC default = #ffee06

color4 = #e0e0e0
#QtPlasmaC default = #26261e

color5 = #808080
#QtPlasmaC default = #b0b0b0

*****/

```

The colors may be expressed in any valid stylesheet format.

The above colors are used for the following widgets. So any custom styling will need to take these into account. The colors shown below are the defaults used in QtPlasmaC along with the color name from the [PARAMETERS Tab](#).

Color	Parameter	Affects
color1 (#ffee06)	Foreground	foreground of jog buttons foreground of latching user buttons foreground of camera/laser buttons foreground of conversational shape buttons background of active conversational shape buttons
color2 (#16160e)	Background	background of latching user buttons background of camera/laser buttons background of G-code editor active line background of conversational shape buttons
color3 (#ffee06)	Highlight	background of active latching user buttons background of active camera/laser buttons foreground of G-code editor cursor
color4 (#36362e)	Alt Background	background of G-code display active line

5.7.14.3 Returning To The Default Styling

The user may return to the default styling at any time by following the following steps:

1. Close QtPlasmaC if open.
2. Delete qtplasmac.qss from the machine config directory.
3. Delete qtplasmac_custom.qss from the machine config directory (if it exists).
4. Open <machine_name>.prefs file.
5. Delete the "[COLOR_OPTIONS]" section.
6. Delete the Custom style line from the '[GUI_OPTIONS]' section.
7. Save the file.

The next time QtPlasmaC is loaded all custom styling will be removed and the default styling will return.

Below is an example of the section to be deleted from <machine_name>.prefs:

```
[COLOR_OPTIONS]
Foreground = #ffee06
Highlight = #ffee06
LED = #ffee06
Background = #16160e
Background Alt = #36362e
Frames = #ffee06
Estop = #ff0000
Disabled = #b0b0b0
Preview = #000000
```

5.7.15 QtPlasmaC Advanced Topics

5.7.15.1 Custom User Buttons

The QtPlasmaC GUI offers user buttons that can be customized by adding commands in the <machine_name>.prefs file.

The number of user buttons varies by display type and resolution as follows:

- 16:9 and 4:3 - Minimum 8, Maximum 20
- 9:16 - Minimum 15, Maximum 20

The user will need to run QtPlasmaC at the desired screen size to determine how many user buttons are available for use.

All <machine_name>.prefs file settings for the buttons are found in the [BUTTONS] section.

Button Names The text that appears on the button is set the following way:

```
n Name = HAL Show
```

Where n is the button number and **HAL Show** is the text.

For text on multiple lines, split the text with a \

```
n Name = HAL\Show
```

If an ampersand is required to be displayed as text then two consecutive ampersands are required:

```
n Name = PIERCE\&&CUT
```

Button Code Buttons can run the following:

1. [External commands](#)
2. [G-code commands](#)
3. [Toggle a HAL pin](#)
4. [Toggle the alignment laser HAL pin](#)
5. [Pulse a HAL pin](#)
6. [Probe test](#)
7. [Ohmic Test](#)
8. [Cut Type](#)
9. [Change consumables](#)
10. [Load a G-code program](#)
11. [Pulse the torch on](#)
12. [Single unidirectional cut](#)
13. [Framing a job](#)
14. [Begin/End a manual cut](#)
15. [Display/Hide an offsets viewer](#)
16. [Load the latest modified ngc file found in a directory](#)

External Commands

To run an external command, the command is preceded by a % character.

```
n Code = %halshow
```

G-code

To run G-code, just enter the code to be run.

```
n Code = G0 X100
```

To run an existing subroutine.

```
n Code = o<the_subroutine> call
```

<machine_name>.ini file variables can be entered by using { } (a space must be placed after the)

```
n Code = G0 X{JOINT_0 HOME} Y1  
n Code = G53 G0 Z[{AXIS_Z MAX_LIMIT} - 1.001]
```

Multiple codes can be run by separating the codes with a \ symbol. The exception is the special commands which are required to be a single command per button.

```
n Code = G0 X0 Y0 \ G1 X5 \ G1 Y5
```

External commands and G-code may be mixed on the same button.

```
n Code = %halshow \ g0x.5y.5 \ %halmeter
```

Toggle HAL Pin

The following code will allow the user to use a button to invert the current state of a HAL bit pin:

```
n Code = toggle-halpin the-hal-pin-name
```

This code is required to be used as a single command and may only control one HAL bit pin per button.

The button colors will follow the state of the HAL pin.

After setting the code, upon clicking, the button will invert colors and the HAL pin will invert pin state. The button will stay "latched" until the button is clicked again, which will return the button to the original colors and the HAL pin to the original pin state.

There are three [External HAL Pins](#) that are available to toggle as an output, the pin names are qtplasmac.ext_out_0, qtplasmac.ext_out_1, and qtplasmac.ext_out_2. HAL connections to these HAL pins need to be specified in a postgui HAL file as the HAL pins are not available until the QtPlasmac GUI has loaded.

It is possible for the user to mark a toggle-halpin button's associated HAL pin as being required to be turned "ON" before **CYCLE START** can be pressed by adding "runcritical" after the HAL pin in the button code.

```
n Code = toggle-halpin the-hal-pin-name runcritical
```

Toggle Alignment Laser HAL Pin

The following code will allow the user to use a button to invert the current state of the alignment laser HAL bit pin:

```
n Code = toggle-laser
```

This code is also able to be used as a multiple command with G-code or external commands but may control only the alignment laser HAL bit pin.

The button colors will follow the state of the alignment laser HAL pin.

After setting the code, upon clicking, the button will invert colors and the alignment laser HAL pin will invert pin state. The button will stay "latched" until the button is clicked again, which will return the button to the original colors and the alignment laser HAL pin to the original pin state.

The following code would allow the user to use a button to invert the current state of the alignment laser HAL bit pin and then move the X and Y axes to the offset for the alignment laser as specified in the <machine_name>.prefs file:

```
n Code = G0 {QTPLASMAC LASER_TOUCHOFF} \ toggle-laser
```

The position of the "toggle-laser" command is not important as it is always the first command actioned regardless of position.

Pulse HAL Pin

The following code will allow the user to use a button to pulse a HAL bit pin for a duration of 0.5 seconds:

```
n Code = pulse-halpin the-hal-pin-name 0.5
```

This code is required to be used as a single command and may only control one HAL bit pin per button.

The pulse duration is specified in seconds, if the pulse duration is not specified then it will default to one second.

The button colors will follow the state of the HAL pin.

After setting the code, upon clicking the button, the button will invert colors, the HAL pin will invert pin state, and the time remaining will be displayed on the button. The button color and the pin state

will stay inverted until the pulse duration timer has completed, which will return the button to the original colors, the HAL pin to the original pin state, and the original button name.

An active pulse can be canceled by clicking the button again.

There are three [External HAL Pins](#) that are available to pulse as an output, the pin names are `qtplasmac.ext_out_0`, `qtplasmac.ext_out_1`, and `qtplasmac.ext_out_2`. HAL connections to these HAL pins need to be specified in a postgui HAL file as the HAL pins are not available until the QtPlasmac GUI has loaded.

Probe Test

QtPlasmaC will begin a probe and when the material is detected, the Z axis will rise to the Pierce Height currently displayed in the MATERIAL section of the [PARAMETERS Tab](#). If the user has "View Material" selected in the GUI SETTINGS section of the [PARAMETERS Tab](#), this value will be displayed in the top left corner of the PREVIEW Window next to **PH:**.

QtPlasmaC will then wait in this state for the time specified (rounded to no decimal places) before returning the Z axis to the starting position. An example of a 6 second delay is below. If there is no time specified then the probe time will default to 10 seconds.

```
n Code = probe-test 6
```

Note

Enabling a user button as a Probe Test button will add an [external HAL pin](#) that may be connected from a pendant etc. HAL connections to this HAL pin needs to be specified in a postgui HAL file as the HAL pin is not available until the QtPlasmac GUI has loaded.

Ohmic Test

QtPlasmaC will enable the Ohmic Probe Enable output signal and if the Ohmic Probe input is sensed, the LED indicator in the SENSOR Panel will light. The main purpose of this is to allow a quick test for a shorted torch tip.

```
n Code = ohmic-test
```

Note

Enabling a user button as an Ohmic Test button will add an [external HAL pin](#) that may be connected from a pendant etc. HAL connections to this HAL pin needs to be specified in a postgui HAL file as the HAL pin is not available until the QtPlasmac GUI has loaded.

Cut Type

This button if selected will toggle between the two [cut types](#), Pierce and Cut (default cutting mode) or Pierce Only.

```
n Code = cut-type
```

Change Consumables

Pressing this button moves the torch to the specified coordinates when the machine is paused to allow the user easy access to change the torch consumables.

Valid entries are `Xnnn Ynnn Fnnn`. Feed Rate (F) is mandatory and at least one of the X or Y coordinates are required.

The X and Y coordinates are in absolute machine coordinates. If X or Y are missing then the current coordinate for that axis will be used.

There are three methods to return to the previous coordinates:

1. Press the **Change Consumables** button again - the torch will return to the original coordinates and the machine will wait in this position for the user to resume the program.
2. Press **CYCLE RESUME** - the torch to return to the original coordinates and the program will resume.
3. Press **CYCLE STOP** - the torch to return to the original coordinates and the program will abort.

```
n Code = change-consumables X10 Y10 F1000
```

Note

Enabling a user button as a Change Consumables button will add an [external HAL pin](#) that may be connected from a pendant etc. HAL connections to this HAL pin needs to be specified in a postgui HAL file as the HAL pin is not available until the QtPlasmac GUI has loaded.

Load

Loading a G-code program from the directory specified by the **PROGRAM_PREFIX** variable in the <machine_name>.ini file (usually ~/linuxcnc/nc_files) is possible by using the following format:

```
n Code = load G-code.ngc
```

If the user's G-code file is located in a sub-directory of the **PROGRAM_PREFIX** directory, it would be accessed by adding the sub-directory name to the beginning of the G-code file name. Example for a sub-directory named **plasma**:

```
n Code = load plasma/G-code.ngc
```

Note that the first "/" is not necessary as it will be added automatically.

Torch Pulse

Pulse the torch on for a predetermined time. The time must be specified in seconds using up to one decimal place. The maximum allowable time is 3 seconds, anything specified above that value will be limited to 3 seconds. An example of a 0.5 second pulse is below. If there is no time specified then it will default to 1 second. Pulse times with more than one decimal place will be rounded to one decimal place.

Pressing the button again during the countdown will cause the torch to be turned off, as will pressing Esc if keyboard shortcuts are enabled in the [PARAMETERS Tab](#).

If the button is released before the countdown is complete then the torch will turn off at countdown completion, holding the button on until after the countdown has completed will cause the torch to remain on until the button has been released.

```
n Code = torch-pulse 0.5
```

Note

Enabling a user button as a Torch Pulse button will add an [external HAL pin](#) that may be connected from a pendant etc. HAL connections to this HAL pin needs to be specified in a postgui HAL file as the HAL pin is not available until the QtPlasmac GUI has loaded.

Single Cut

Run a single unidirectional cut. This utilises the automatic [Single Cut](#) feature.

```
n Code = single-cut
```

Framing

Framing is the ability to move the torch around the perimeter of a rectangle that encompasses the bounds of the current job.

The laser enable HAL pin (qtplasmac.laser_on) will be turned on during the framing moves and any X/Y offsets for the laser pointer in the <machine_name>.prefs file will also be applied to the X/Y motion. After the framing motion is completed, the torch will move to the X0 Y0 position to clear any applied laser offsets and qtplasmac.laser_on will be turned off.

Upon starting a Framing cycle, it is important to note that by default the Z axis will be moved to a height of [AXIS_Z]MAX_LIMIT - 5mm (0.2") before X/Y motion begins.

The velocity for the XY movements of the Framing motion can be specified so that Framing motion always occurs at a set velocity. This can be achieved by adding the feed rate (F) as the last portion of the button code. If the feed rate is omitted from the button code, framing motion velocity will default to the feed rate for the currently selected material.

The following GUI buttons and Keyboard Shortcuts (if enabled in the [PARAMETERS Tab](#)) are valid during Framing motion:

1. Pressing **CYCLE STOP** or the ESC [keyboard shortcut](#) - Stops Framing motion.
2. Pressing **CYCLE PAUSE** or the SPACE BAR [keyboard shortcut](#)- Pauses Framing motion.
3. Pressing **CYCLE RESUME** or the CTRL+r [keyboard shortcut](#)- Resumes paused Framing motion.
4. Changing the **FEED SLIDER** or any of the CTRL+0-9 [keyboard shortcuts](#) - Slows the feed rate.

Note

IF THE FEED RATE IS CHANGED FOR THE FRAMING MOTION, IT WILL BE NECESSARY TO RETURN THE FEED SLIDER TO 100% BEFORE PRESSING CYCLE START AND CUTTING THE LOADED JOB.

```
n Code = framing
```

It is possible for the user to omit the initial default Z movement and run the framing sequence at the current Z height by adding "usecurrentzheight" after "framing".

```
n Code = framing usecurrentzheight
```

To specify a feed rate:

```
n Code = framing F100
```

or:

```
n Code = framing usecurrentzheight F100
```

Enabling a user button as a framing button will add an [external HAL pin](#) that may be connected from a pendant etc. HAL connections to this HAL pin needs to be specified in a postgui HAL file as the HAL pin is not available until the QtPlasmac GUI has loaded.

Manual Cut

Manual Cut functions identically to the **F9** button to begin or end a [manual cut](#).

```
n Code = manual-cut
```

Offset Viewer

This allows the showing/hiding of an offset viewing screen that displays all machine offsets. All relative offsets can be edited and the G54 ~ G59.3 work system coordinates are able to be given custom names.

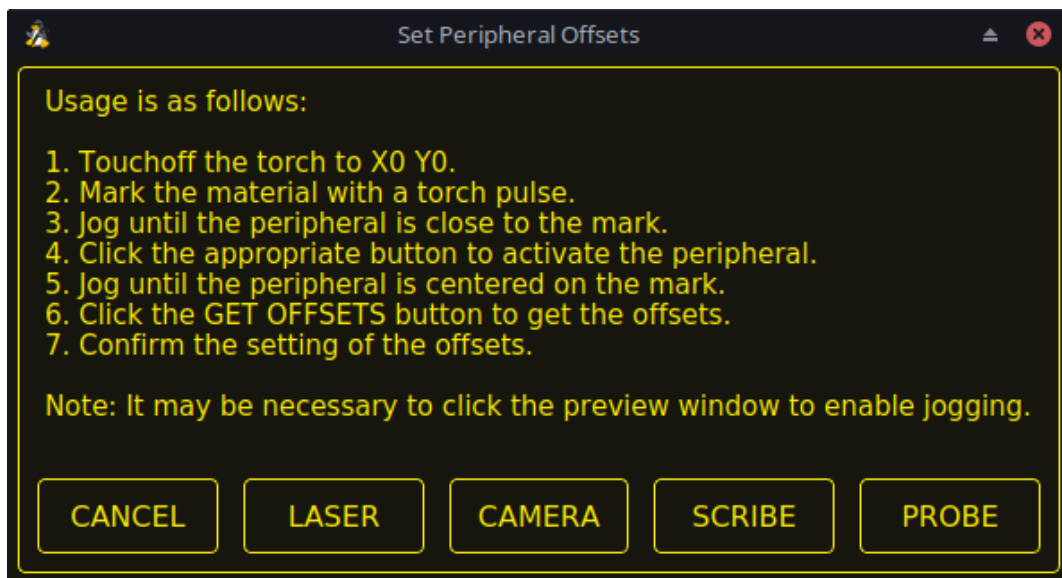
```
n Code = offsets-view
```

[underline]*Load Latest File

this allows the loading of the last modified file in a directory. The directory name is optional and if omitted will default to the last directory a file was loaded from.

```
n Code = latest-file /home/me/linuxcnc/nc_files/qtplasmac-test
```

5.7.15.2 Peripheral Offsets (Laser, Camera, Scribe, Offset Probe)



Use the following sequence to set the offsets for a laser, camera, scribe, or offset probe:

1. Place a piece of scrap material under the torch.
2. The machine must be homed and idle before proceeding.
3. Open the [PARAMETERS Tab](#) (or SETTINGS tab for 4x3 GUI).
4. Click the SET OFFSETS button which opens the Set Peripheral Offsets dialog.
5. Click the X0Y0 button to set the torch position to zero.
6. Make a mark on the material by one of:
 - a. Jog the torch down to pierce height then pulse the torch on to make a dimple in the material.
 - b. Place marking dye on the torch shield then jog the torch down to mark the material.
7. Raise the Z axis so the torch and peripheral are clear of the material
8. Jog the X/Y axes so that the peripheral is close to the mark from the torch.
9. Click the appropriate button to activate the peripheral.
10. Jog the X/Y axes so that the peripheral is centered in the mark from the torch.
11. Click the GET OFFSETS button to get the offsets and a confirmation dialog will open.
12. Click SET OFFSETS and the offsets will now be saved.

Canceling may be done at any stage by pressing the CANCEL button which will close the dialog and no changes will be saved.

Note

It may be necessary to click the preview window to enable jogging. By following the above procedure the offsets are available for use immediately and no restart of LinuxCNC is required.

5.7.15.3 Keep Z Motion

By default, QtPlasmaC will remove all Z motion from a loaded G-code file and add an initial Z movement to bring the torch near the top of travel at the beginning of the file. If the user wishes to use their table with a marker, a drag knife, diamond scribe, etc. mounted in the torch holder, QtPlasmaC has the ability to retain the Z movements when executing a program by adding the following command in a G-code file:

```
#<keep-z-motion> = 1
```

Omitting this command, or setting this value to anything but 1 will cause QtPlasmaC to revert to the default behavior of stripping all Z motion from a loaded G-code file and making an initial Z movement to bring the torch near the top of travel at the beginning of the file.

5.7.15.4 External HAL Pins

QtPlasmaC creates some HAL pins that may be used to connect a momentary external button or pendant etc.

HAL connections to these HAL pins need to be specified in a postgui HAL file as the HAL pins are not available until the QtPlasmaC GUI has loaded.

The following HAL bit pins are always created. The HAL pin has the identical behaviour of the related QtPlasmaC GUI button.

User Button Function	HAL Pin	GUI Function
Toggle machine power	qtplasmac.ext_power	POWER
Run the loaded G-code program	qtplasmac.ext_run	CYCLE START
Pause/Resume the loaded G-code program	qtplasmac.ext_pause	CYCLE PAUSE
Abort the loaded G-code program	qtplasmac.ext_abort	CYCLE STOP
Touchoff X & Y axes to zero	qtplasmac.ext_touchoff	X0Y0
Use a laser to set an origin with or without rotation	qtplasmac.ext_laser_touchoff	LASER
Run/Pause/Resume the loaded G-code program	qtplasmac.ext_run_pause	CYCLE START, CYCLE PAUSE, CYCLE RESUME in sequence
Torch height override plus	qtplasmac.ext_height_override_plus	OVERRIDE
Torch height override minus	qtplasmac.ext_height_override_minus	OVERRIDE -
Torch height override reset	qtplasmac.ext_height_override_reset	OVERRIDE RESET TO 0.00
Torch height override scale	qtplasmac.ext_height_override_scale	OVERRIDE SCALE
Toggle jogging speed between fast and slow	qtplasmac.ext_jog_slow	JOGGING FAST/SLOW
Toggle THC enable	qtplasmac.ext_thc_enable	THC ENABLE
Toggle torch enable	qtplasmac.ext_torch_enable	TORCH ENABLE
Toggle corner Lock enable	qtplasmac.ext_cornerlock_enable	VELOCITY ANTI DIVE ENABLE

User Button Function	HAL Pin	GUI Function
Toggle kerf cross enable	qtplasmac.ext_kerfcross_enable	WELD ANTI DIVE ENABLE
Toggle use auto volts	qtplasmac.ext_auto_volts_enable	Auto VOLTS
Toggle ohmic probe enable	qtplasmac.ext_ohmic_probe_enable	OHMIC ENABLE
Toggle mesh mode	qtplasmac.ext_mesh_mode	MESH MODE
Toggle arc ignore OK	qtplasmac.ext_ignore_arc_ok	IGNORE OK
Forward along the programmed path	qtplasmac.ext_cutrec_fwd	CUT RECOVERY FWD
Reverse along the programmed path	qtplasmac.ext_cutrec_rev	CUT RECOVERY REV
Cancel any Cut Recovery movement	qtplasmac.ext_cutrec_cancel	CUT RECOVERY CANCEL MOVE
Move up	qtplasmac.ext_cutrec_n	CUT RECOVERY arrow up
Move down	qtplasmac.ext_cutrec_s	CUT RECOVERY arrow down
Move right	qtplasmac.ext_cutrec_e	CUT RECOVERY arrow right
Move left	qtplasmac.ext_cutrec_w	CUT RECOVERY arrow left
Move up-right	qtplasmac.ext_cutrec_ne	CUT RECOVERY arrow up-right
Move up-left	qtplasmac.ext_cutrec_nw	CUT RECOVERY arrow up-left
Move down-right	qtplasmac.ext_cutrec_se	CUT RECOVERY arrow down-right
Move down-left	qtplasmac.ext_cutrec_sw	CUT RECOVERY arrow down-left

The following HAL pins which allow the use of an MPG to control height override are always created.

Function	HAL Pin
Enable MPG height control	qtplasmac.ext_height_ovr_count_enable
MPG height change	qtplasmac.ext_height_ovr_counts

The following HAL bit pins are only created if the function is specified in a [custom user button](#). The HAL pin has the identical behaviour of the related custom user button.

User Button Function	HAL Pin
Probe Test	qtplasmac.ext_probe
Torch Pulse	qtplasmac.ext_pulse
Ohmic Test	qtplasmac.ext_ohmic
Change Consumables	qtplasmac.ext_consumables
Framing	qtplasmac.ext_frame_job

The following HAL bit output pins are always created and can be used by either the [Toggle HAL Pin](#) or [Pulse HAL Pin](#) custom user buttons to change the state of an output.

HAL Pin
qtplasmac.ext_out_0
qtplasmac.ext_out_1
qtplasmac.ext_out_2

5.7.15.5 Hide Program Buttons

If the user has external buttons and/or a pendant that emulates any of the program buttons, CYCLE START, CYCLE PAUSE, or CYCLE STOP then it is possible to hide any or all of these GUI program buttons by adding the following code to the [GUI_OPTIONS] section of the <machine_name>.prefs file:

```
Hide run = True
Hide pause = True
Hide abort = True
```

For the 16:9 or 4:3 GUIs, the hiding of each of these GUI buttons will expose two more custom user buttons in the GUI.

5.7.15.6 Tuning Void Sensing

Void sensing may be further tuned by adjusting the required number of consecutive times that the threshold has been exceeded before applying the void lock to the THC.

The HAL pin for adjusting this is named `plasmac.kerf-errors-max` and the default value is 2. To change this value, add the pin and the required value to the `custom.hal` file. The following example would set the required number of consecutive times to exceed the threshold to 3:

```
setp plasmac.kerf-error-max 3
```

5.7.15.7 Lost Arc Delay

Some plasma power sources/machine configurations may lose the Arc OK signal either momentarily during a cut, or permanently near the end of a cut causing QtPlasmaC to pause the program and report a "valid arc lost" error. There is a HAL pin named **`plasmac.arc-lost-delay`** that may be used to set a delay (in seconds) that will prevent a paused program/error if the lost Arc OK signal is regained, or the **M5** command is reached before the set delay period expires. It is important to note that the THC will be disabled and locked at the cutting height at the time the Arc OK signal was lost.

The following code would set a delay of 0.1 seconds:

```
setp plasmac.arc-lost-delay 0.1
```

It is recommended that the user set this pin in the `custom.hal` file.

This setting should only be used if the user experiences the above symptoms. It should also be noted that the user could use the appropriate [Ignore Arc OK](#) G-code commands to achieve a similar result.

5.7.15.8 Tuning Mode 0 Arc OK

Mode 0 Arc OK relies on the arc voltage to set the Arc OK signal. This is accomplished by sampling the arc voltage every servo thread cycle. There needs to be a specified number of consecutive samples all within a specified threshold for the Arc OK signal to be set. These voltages are also required to be within a specified range.

There are two settings in the [PARAMETERS Tab](#) for setting the range, these are:

- **OK HIGH VOLTS** which is the upper value of the voltage range. The default is 250V.
- **OK LOW VOLTS** which is the lower value of the voltage range. The default is 60V.

Both of these values may be changed by direct entry or by the use of the increment/decrement buttons. There are also two HAL pins that have been provided to allow the user to tune the set point. These HAL pins are:

- **`plasmac.arc-ok-counts`** which is the number of consecutive readings within the threshold that are required to set the Arc OK signal. The default is 10.

- **plasmac.arc-ok-threshold** which is the maximum voltage deviation that is allowed for a valid voltage to set the Arc OK signal. The default is 10.

The following example would set the number of valid consecutive readings required to 6:

```
setp plasmac.arc-ok-counts 6
```

5.7.15.9 Zero Window

Small fluctuations in the arc voltage displayed while the machine is at idle are possible depending on many different variables (electrical noise, incorrect THCAD tuning, etc.). After all contributing factors have been mitigated, if a small fluctuation still exists it is possible to eliminate it by widening the voltage window for which QtPlasmaC will display 0v.

The pin for adjusting this value is named `plasmac.zero-window` and the default value is set to 0.1. To change this value, add the pin and the required value to the `custom.hal` file. The following example would set the voltage window to be displayed as 0v from -5v to +5v:

```
setp plasmac.zero-window 5
```

5.7.15.10 Max Offset

Max Offset is the distance (in millimeters) away from the `Z MAX_LIMIT` that QtPlasmaC will allow the Z axis to travel while under machine control.

The pin for adjusting this value is named `plasmac.max-offset` and the default value (in millimeters) is set to 5. To change this value, add the pin and the required value to the `custom.hal` file. It is not recommended to use values less than 5mm as offset overrun may cause unforeseen issues.

The following example would set the distance from `Z MAX_LIMIT` to 10 mm:

```
setp plasmac.max-offset 10
```

5.7.15.11 Enable Tabs During Automated Motion

By default, all tabs except the [MAIN Tab](#) are disabled during automated motion. It is possible for every tab but the [CONVERSATIONAL Tab](#) to be enabled during automated motion by setting the following HAL pin True:

```
setp qtplasmac.tabs_always_enabled 1
```



Warning

It is the responsibility of the operator to ensure that the machine is equipped with a suitable, working hardware ESTOP. If using only a touchscreen to navigate the QtPlasmaC GUI, there is no way to stop automated machine motion on any tab but the MAIN tab.

5.7.15.12 Override Jog Inhibit Via Z+ Jog

It is possible to override the jog inhibit by using the GUI or keyboard to jog in the Z+ direction rather than checking the Override Jog box on the [PARAMETERS Tab](#).

This is done by changing the following preference to **True** in the **[GUI_OPTIONS]** of the `<machine_name>.prefs` file in the `<machine_name>` folder:

```
Override jog inhibit via Z+
```

5.7.15.13 QtPlasmaC State Outputs

The plasmac HAL component has a HAL pin named **plasmac.state-out** which can be used to interface with user-coded components to provide the current state of the component.

The different states QtPlasmaC could encounter are as follows:

State	Name	Description
0	IDLE	idle and waiting for a start command
1	PROBE_HEIGHT	move down to probe height
2	PROBE_DOWN	probe down until material sensed
3	PROBE_UP	probe up until material not sensed, this sets the zero height
4	ZERO_HEIGHT	not used at present
5	PIERCE_HEIGHT	move up to pierce height
6	TORCH_ON	turn the torch on
7	ARC_OK	wait until arc ok detected
8	PIERCE_DELAY	wait for pierce delay time
9	PUDDLE_JUMP	xy motion begins, move to puddle jump height
10	CUT_HEIGHT	move to cut height
11	CUT_MODE_01	cutting in either mode 0 or mode 1
12	CUT_MODE_2	cutting in mode 2
13	PAUSE_AT_END	pause motion at end of cut
14	SAFE_HEIGHT	move to safe height
15	MAX_HEIGHT	move to maximum height
16	END_CUT	end the current cut
17	END_JOB	end the current job
18	TORCHPULSE	a torch pulse is active
19	PAUSED_MOTION	cut recovery motion is active while paused
20	OHMIC_TEST	an ohmic test is active
21	PROBE_TEST	a probe test is active
22	SCRIBING	a scribing job is active
23	CONSUMABLE_CHANGE_ON	Give On consumable change coordinates
24	CONSUMABLE_CHANGE_OFF	Give Off consumable change coordinates
25	CUT_RECOVERY_ON	cut recovery is active
26	CUT_RECOVERY_OFF	cut recovery is deactivated

The DEBUG state is for testing purposes only and will not normally be encountered.

5.7.15.14 QtPlasmaC Debug Print

The plasmac HAL component has a HAL pin named **plasmac.debug-print** which if set to 1 (true) will print to terminal every state change as a debug aid.

5.7.15.15 Hypertherm PowerMax Communications

Communications can be established with a Hypertherm PowerMax plasma cutter that has a RS485 port. This feature enables the setting of **Cut Mode**, **Cutting Amperage** and **Gas Pressure** automatically from the **Cut Parameters** of the material file. In addition, the user will be able to view the PowerMax's **Arc On Time** in hh:mm:ss format on the [STATISTICS Tab](#).

If **Gas Pressure** is set to Zero then the PowerMax will automatically calculate the required pressure from the **Cut Mode**, **Cut Current**, torch type, and torch length.

Changing the cutting mode will set the gas pressure to zero causing the machine to use its automatic gas pressure mode.

The maximum and minimum values of these parameters are read from the plasma cutter and the related spin-buttons in the Cut Parameters are then limited by these values. Gas pressure cannot be changed from zero until communications have been established.

This feature is enabled by setting the correct port name for the PM_PORT variable in the [POWERMAX] section of the <machine_name>.prefs file. If the PM_PORT variable is not set in the <machine_name>.prefs file then the widgets associated with this feature will not be visible.

Example showing enabling the Hypertherm PowerMax Communications on USB0:

```
[POWERMAX]
Port = /dev/ttyusb0
```

If the user is unsure of the name of the port, there is a python script in the configuration directory that will show all available ports and can also be used to test communications with the plasma unit prior to enabling this feature in the QtPlasmaC GUI.

To use the test script follow these instructions:

For a package installation (Buildbot) enter the following command in a terminal window:

```
pmx485-test
```

For a run in place installation enter the following two commands in a terminal window:

```
source ~/linuxcnc-dev/scripts/rip-environment
pmx485-test
```

The gas pressure units display (psi or bar) is determined by the data received during initial setup of the communication link and is then shown next to the Gas Pressure setting in the MATERIAL section of the [PARAMETERS Tab](#).

The PowerMax machine will go into remote mode after communications have been established and may only be controlled remotely (via the QtPlasmaC GUI) at this point. The connection can be validated by observing the PowerMax display.

To switch the PowerMax back to local mode the user can either:

1. Disable PowerMax Comms from the [MAIN Tab](#)
2. Close LinuxCNC which will put the PowerMax into local mode during shutdown.
3. Turn the PowerMax off for 30 seconds and then power it back on.

Tip

If PowerMax communications is active then selecting [Mesh Mode](#) will automatically select CPA mode on the PowerMax unit.

Note

To use the PowerMax communications feature it is necessary to have the python pyserial module installed.

If pyserial is not installed an error message will be displayed.

To install pyserial, enter the following command into a terminal window:

```
sudo apt install python-serial
```

A typical [connection diagram](#) is shown in the appendix of this document as well as confirmed working interfaces.

5.7.16 Internationalisation

It is possible to create translation files for QtPlasmaC to display in the language of the current locale.

To create and or edit a translation file requires that LinuxCNC has been installed as run in place.

The following assumes that the linuxcnc git directory is `~/linuxcnc-dev`.

The `$` indicates a terminal prompt.

All language files are kept in `~/linuxcnc-dev/share/screens/qtplasmac/languages`.

The `qtplasmac.py` file is a python version of the GUI file used for translation purposes.

The `.ts` files are the translation source files for the translations. These are the files that require creating/editing for each language.

The `.qm` files are the compiled translation files used by `pyqt`.

The language is determined by an underscore plus the first two letters of the locale, for example if an Italian translation was being done then it would be `"_it"`. It will be referred to as `"_xx"` in this document, so `"qtplasmac_xx.ts"` in this document would actually be `"qtplasmac_it.ts"` for an Italian translation.

The default locale for QtPlasmaC is `"_en"` which means that any translation files created as `"qtplasmac_en.*"` will not be used for translations.

If any of the required utilities (`pyuic5`, `pylupdate5`, `linguist`) are not installed then the user will need to install `pyqt5-dev-tools`:

```
$ sudo apt install pyqt5-dev-tools
```

Change to the languages directory:

```
$ cd ~/linuxcnc-dev/share/qtvcpl/screens/qtplasmac/languages
```

If any text changes have been made to the GUI then run the following to update the GUI python file:

```
$ pyuic5 ../qtplasmac.ui > qtplasmac.py
```

The user can either create a new translation source file for a non-existing language translation or modify an existing translation source file due to changes being made to some text in a QtPlasmaC source file. If modifying an existing translation that has had no source file changes then this step is not required.

Create or edit a `.ts` file:

```
$ langfile xx
```

Note

this command is a script which runs the following: `$ pylupdate5 .py ../.py .././.././../lib/python/qtvcpl/lib/qtplasmac/*.py -ts qtplasmac_xx.ts`

The editing of the translation is done with the `linguist` application:

```
$ linguist
```

Then open the required translation file.

It is not necessary to provide a translation for every text string, if no translation is specified for a string then the original string will be used in the application. The user needs to be careful with the length of strings that appear on widgets as space is limited. If possible try to make the translation no longer than the original.

When editing is complete save the file:

```
File > Save
```

Then create the .qm file:

```
File > Release
```

Then create links to the compiled .qm file for the other QtPlasmaC GUIs.

```
$ ln -s qtplasmac_en.qm ../../qtplasmac_4x3/languages/  
$ ln -s qtplasmac_en.qm ../../qtplasmac_9x16/languages/
```

QtPlasmaC will be translated to the language of the current locale on the next start so long as a .qm file exists in that language.

5.7.17 Appendix

5.7.17.1 Example Configurations

There are example configuration files which use the QtPlasmaC GUI to simulate plasma cutting machines.

They can be found in the LinuxCNC chooser under: Sample Configurations -> by_machine -> qtplasmac

Three versions are available in both metric and imperial units:

1. qtplasmac_l - 16:9 format, minimum resolution 1366x768
2. qtplasmac_p - 9:16 format, minimum resolution 786x1366
3. qtplasmac_s - 4:3 format, minimum resolution 1024x768

Each sample configuration includes a popup control panel to simulate various inputs to the GUI such as:

1. ARC VOLTAGE
2. OHMIC SENSE
3. FLOAT SWITCH
4. BREAKAWAY SWITCH
5. ESTOP

5.7.17.2 NGC Samples

There are some sample G-code files in the ~/linuxcnc/nc_files/examples/plasmac directory.

Description	Code
-------------	------

5.7.17.3 QtPlasmaC Specific G-codes

Description	Code
Begin cut	M3 \$0 S1
End cut	M5 \$0
Begin scribe	M3 \$1 S1
End scribe	M5 \$1
Begin center spot	M3 \$2 S1
End center spot	M5 \$2
End all the above	M5 \$-1
Select a material	M190 Pn n denotes the material number
Wait for material change confirmation	M66 P3 L3 Qn n is delay time (in seconds) This value may need to be increased for very large material files
Set feed rate from material	F#<_hal[plasmac.cut-feed-rate]>
Enable Ignore Arc OK	M62 P1 (synchronized with motion) M64 P1 (immediate)
Disable Ignore Arc OK	M63 P1 (synchronized with motion) M65 P1 (immediate)
Disable THC	M62 P2 (synchronized with motion) M64 P2 (immediate)
Enable THC	M63 P2 (synchronized with motion) M65 P2 (immediate)
Disable Torch	M62 P3 (synchronized with motion) M64 P3 (immediate)
Enable Torch	M63 P3 (synchronized with motion) M65 P3 (immediate)
Set velocity to a percentage of feed rate	M67 E3 Qn (synchronized with motion) M68 E3 Qn (immediate) n is the percentage to set 10 is the minimum, below this will be set to 100% 100 is the maximum, above this will be set to 100% It is recommended to have M68 E3 Q0 in both the preamble and postamble
Cutter compensation - left of path	G41.1 D#<_hal[plasmac_run.kerf-width-f]>
Cutter compensation - right of path	G42.1 D#<_hal[plasmac_run.kerf-width-f]>
Cutter compensation off	G40 Note that M62 through M68 are invalid while cutter compensation is on
Cut holes at 60% feed rate	#<holes> = 1 for holes less than 32 mm (1.26") diameter
Cut holes at 60% feed rate, turn torch off at hole end, continue hole path for over cut	#<holes> = 2 for holes less than 32 mm (1.26") diameter over cut length = 4mm (0.157")
Cut holes and arcs at 60% feed rate	#<holes> = 3 for holes less than 32 mm (1.26") diameter for arcs less than 16 mm (0.63") radius

Description	Code
Cut holes and arcs at 60% feed rate, turn torch off at hole end, continue hole path for over cut	#<holes> = 4 for holes less than 32 mm (1.26") diameter for arcs less than 16 mm (0.63") radius over cut length = 4 mm (0.157")
Specify hole diameter for #<holes> = 1-4	#<h_diameter> = n (n is the diameter, use the same units system as the rest of the G-code file)
Specify hole velocity for #<holes>=1-4	#<h_velocity> = n (n is the percentage, set the percentage of the current feed rate)
Specify over cut length	#<oclength> = n (n is the length, use the same units system as the rest of the G-code file)
Specify pierce-only mode	#<pierce-only> = n (n is the mode, 0=normal cut mode, 1=pierce only mode)
Create or edit materials. options: 0 - Create temporary default 1 - Add if not existing 2 - Overwrite if existing else add new	mandatory parameters: (o=<option>, nu=<nn>, na=<ll>, ph=<nn>, pd=<nn>, ch=<nn>, fr=<nn>) optional parameters: (kw=<nn>, th=<nn>, ca=<nn>, cv=<nn>, pe=<nn>, gp=<nn>, cm=<nn>, jh=<nn>, jd=<nn>)
Keep Z Motion	#<keep-z-motion> = 1

5.7.17.4 QtPlasmaC G-code Examples

Description	Example
Select material and do a normal cut	M190 P3 M66 P3 L3 Q1 F#<_hal[plasmac.cut-feed-rate]> M3 \$0 S1 . . M5 \$0
Set velocity to 100% of CutFeedRate	M67 E3 Q0 or M67 E3 Q100
Set velocity to 60% of CutFeedRate	M67 E3 Q60
Set velocity to 40% of CutFeedRate	M67 E3 Q40
Cut a hole with 60% reduced speed using velocity setting	G21 (metric) G64 P0.05 M52 P1 (allow paused motion) F#<_hal[plasmac.cut-feed-rate]> G0 X10 Y10 M3 \$0 S1 (start cut) G1 X0 M67 E3 Q60 (reduce feed rate to 60%) G3 I10 (the hole) M67 E3 Q100 (restore feed rate to 100%) M5 \$0 (end cut) G0 X0 Y0 M2 (end job)

Description	Example
Cut a hole with 60% reduced speed using the #<holes> command	G21 (metric) G64 P0.05 M52 P1 (allow paused motion) #<holes> = 1 (velocity reduction for holes) F#<_hal[plasmac.cut-feed-rate]> G0 X10 Y10 M3 \$0 S1 (start cut) G1 X0 G3 I10 (the hole) M5 \$0 (end cut) G0 X0 Y0 M2 (end job)
Cut a hole with over cut using torch disable	G21 (metric) G64 P0.05 M52 P1 (allow paused motion) F#<_hal[plasmac.cut-feed-rate]> G0 X10 Y10 M3 \$0 S1 (start cut) G1 X0 M67 E3 Q60 (reduce feed rate to 60%) G3 I10 (the hole) M62 P3 (turn torch off) G3 X0.8 Y6.081 I10 (continue motion for 4mm) M63 P3 (allow torch to be turned on) M67 E3 Q0 (restore feed rate to 100%) M5 \$0 (end cut) G0 X0 Y0 M2 (end job)
Cut a hole with over cut using the #<holes> command	G21 (metric) G64 P0.05 M52 P1 (allow paused motion) #<holes> = 2 (over cut for holes) F#<_hal[plasmac.cut-feed-rate]> G0 X10 Y10 M3 \$0 S1 (start cut) G1 X0 G3 I10 (the hole) M5 \$0 (end cut) G0 X0 Y0 M2 (end job)
Cut a hole with 6.5mm over cut using the #<holes> command	G21 (metric) G64 P0.05 M52 P1 (allow paused motion) #<holes> = 2 (over cut for holes) <oclength> = 6.5 (6.5mm over cut length) F<_hal[plasmac.cut-feed-rate]> G0 X10 Y10 M3 \$0 S1 (start cut) G1 X0 G3 I10 (the hole) M5 \$0 (end cut) G0 X0 Y0 M2 (end job)

Description	Example
Select scribe and select torch at end of scribing	<pre> . . M52 P1 (paused motion on) F#<_hal[plasmac.cut-feed-rate]> T1 M6 (select scribe) G43 H0 (apply offsets) M3 \$1 S1 (start plasmac with scribe) . . T0 M6 (select torch) G43 H0 (apply offsets) G0 X0 Y0 (parking position) M5 \$1 (end) </pre>
Hole center spotting.	<pre> (Requires a small motion command or nothing happens) G21 (metric) F999999 (high feed rate) G0 X10 Y10 M3 \$2 S1 (spotting on) G91 (relative distance mode) G1 X0.000001 G90 (absolute distance mode) M5 \$2 (spotting off) G0 X0 Y0 G90 M2 </pre>
Create temporary default material	(o=0, nu=2, na=5mm Mild Steel 40A, ph=3.1, pd=0.1, ch=0.75, fr=3000)
Edit material, if not existing create a new one	(o=2, nu=2, na=5mm Mild Steel 40A, ph=3.1, pd=0.1, ch=0.75, fr=3000, kw=1.0)

5.7.17.5 Mesa THCAD

The Mesa THCAD is a common way of obtaining the arc voltage from a plasma cutter and is also useful for ohmic sensing of the material during probing. The THCAD may be used for parallel port configurations as well as configurations using Mesa Electronics hardware. The THCAD is available in three different models, THCAD-5, THCAD-10, and THCAD-300.

There is a mode jumper on each THCAD card which should be set to **UNIPOLAR**

There is a frequency divider jumper on each THCAD card which should be set according to the hardware type:

Input Device	Recommended Setting
Parallel Port with very low latency	F/32
Parallel Port recommended starting point	F/64
Parallel Port with higher latency, or when cutting thick material	F/128

Input Device	Recommended Setting
Mesa Card	F/32

This value is required to be entered into PNcConf during installation.

Note

If using a parallel port it may be necessary for the user to adjust the jumper setting and the subsequent scaling values on the [Parameters Tab](#) to achieve optimal results. Symptoms may include random torch raises or dives during otherwise stable cutting. Halscope plots may be useful in diagnosing these issues.

Located on the rear of the THCAD is a calibration sticker showing:

THCAD-nnn

0V 121.1 kHz
5V 925.3 kHz

or similar values, these values are required to be entered into PNcConf during installation.

PNcConf has entries for all required THCAD parameters and will calculate and configure any required settings. The calculations used are as follows:

Voltage Scale

$$vs = r / ((f - z) / d / v)$$

Voltage Offset

$$vo = z / d$$

r = divider ratio (see below).

f = full scale value from calibration sticker.

z = 0V value from calibration sticker.

d = value from jumper above.

v = full scale voltage of THCAD

Divider Ratio

THCAD-5 or THCAD-10

If connecting to a plasma CNC port then the divider ratio is selected from the plasma machine. A common ratio used is 20:1

If connecting to the plasma machines full arc voltage then a common setup for a THCAD-10 is to use a 1 Megohm resistor from arc negative to THCAD negative and a 1 Megohm resistor from arc positive to THCAD positive. The divider ratio is obtained by:

$$r = (\text{total_resistance} + 100000) / 100000$$

THCAD-300

$$r = 1$$

**Important**

IF THE USER IS USING A HF START PLASMA POWER SUPPLY THEN EACH OF THESE RESISTANCES SHOULD BE MADE UP OF SEVERAL HIGH VOLTAGE RESISTORS.

**Caution**

IF THE USER IS USING A HF START PLASMA POWER SUPPLY THEN OHMIC SENSING IS NOT RECOMMENDED.

Note

These values can be calculated by using [this online calculator](#).

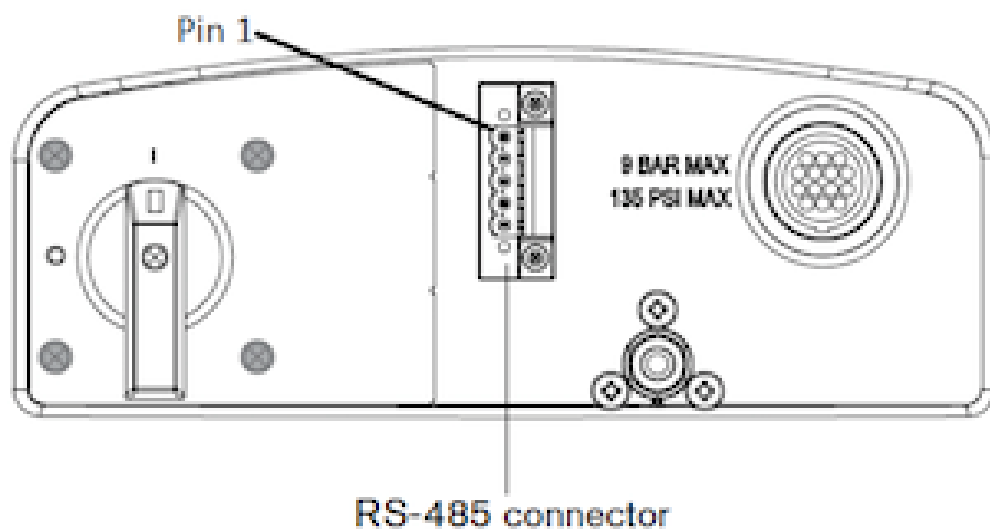
Note

There is a [lowpass filter](#) available which may be useful if using a THCAD and there is a lot of noise on the returned arc voltage.

5.7.17.6 RS485 Connections

Hypertherm RS485 Wiring Diagram (wire colors inside the Hypertherm in parentheses):

Connection at Machine Pin #	Connection at Breakout Board
1 - Tx+ (Red)	->RXD+
2 - Tx- (Black)	->RXD-
3 - Rx+ (Brown)	->T/R+
4 - Rx- (White)	->T/R-
5 - GND (Green)	->GND



RS485 interfaces that are known to work:

DTECH DT-5019 USB to RS-485 converter adapter:



The following is necessary to convert a motherboard Serial connection or Serial card (RS232) to RS485:

DTECH RS-232 to RS-485 converter:



Serial card example (Sunnix SER5037A PCI Card shown with Breakout Board):

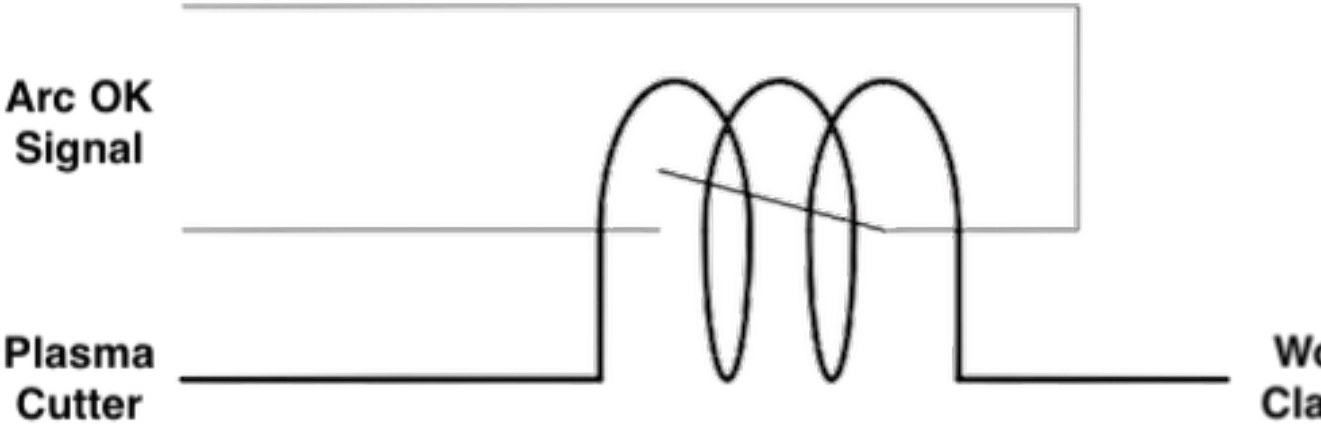


5.7.17.7 Arc OK With A Reed Relay

An effective and very reliable method of obtaining an Arc OK signal from a plasma power supply without a CNC port is to mount a reed relay inside a non-conductive tube and wrap and secure three turns of the work lead around the tube.

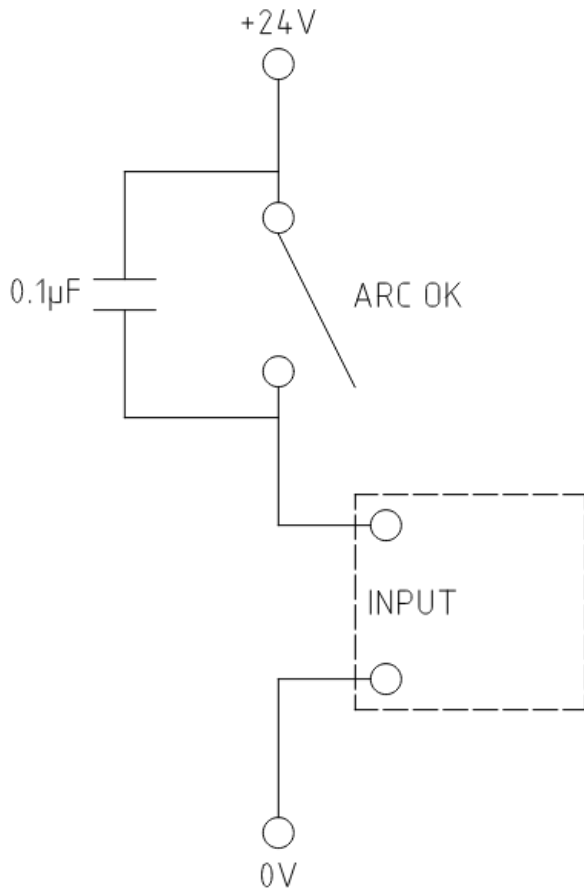
This assembly will now act as a relay that will switch on when current is flowing through the work lead which only occurs when a cutting arc has been established.

This will require that QtPlasmaC be operated in Mode 1 rather than Mode 0. See the [QtPlasmaC Modes](#) sections for more information.

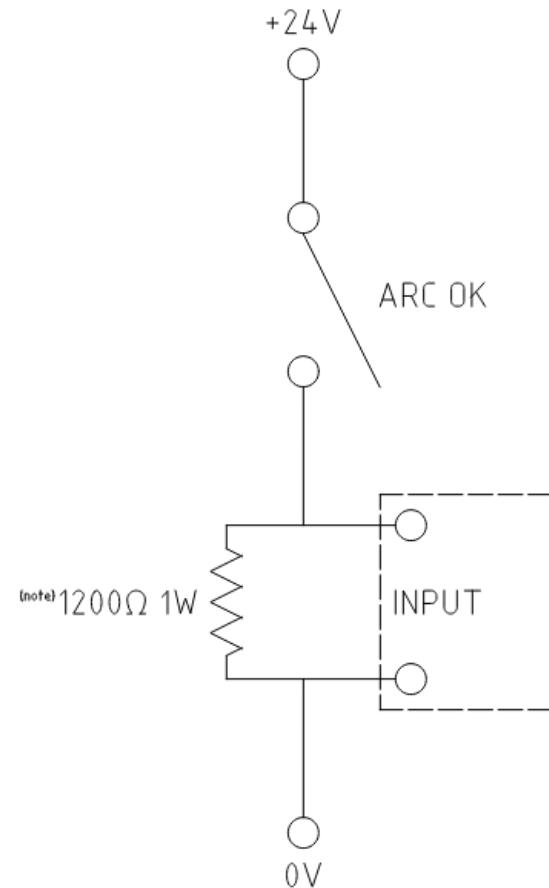


5.7.17.8 Contact Load Schematics

Capacitor Discharge Method



Resistor Wetting Method



Note:

The resistor value needs to be determined from the manufacturer's specifications.

The resistor shown is calculated for Hypertherm 65.

A full description is at [Contact Load](#)

5.7.18 Known Issues

5.7.18.1 Keyboard Jogging

There is a known issue with some combinations of hardware and keyboards that may affect the autorepeat feature of the keyboard and will then affect keyboard jogging by intermittent stopping and starting during jogging. This issue can be prevented by disabling the Operating System's autorepeat feature for all keys. QtPlasmaC uses this disabling feature by default for all keys only when the [MAIN](#)

[Tab](#) is visible, with the following exceptions when autorepeat is allowed with the [MAIN Tab](#) visible: G-code editor is active, MDI is active. When QtPlasmaC is shut down, the Operating System's autorepeat feature will be enabled for all keys.

If the user wishes to prevent QtPlasmaC from changing the Operating System's autorepeat settings, enter the following in the [GUI_OPTIONS] section of the <machine_name>.prefs file:

```
Autorepeat all == True
```

This issue does not affect any jogging using the GUI jog buttons.

Note

Disconnecting and reconnecting a keyboard during an active QtPlasmaC session will cause the autorepeat feature to re-enable itself automatically which may cause intermittent stopping and starting during jogging. The user must restart QtPlasmaC to disable the autorepeat feature again.

5.7.19 Support

Online help and support is available from the [PlasmaC section](#) of the [LinuxCNC Forum](#).

The user can create a compressed file containing the complete machine configuration to aid in fault diagnosis by pressing following the directions in the [backup](#) section. The resulting file is suitable for attaching to a post on the LinuxCNC Forum to help the community diagnose specific issues.

5.8 MDRO GUI

5.8.1 Introduction

MDRO is a simple graphical front-end for LinuxCNC providing a display of data from Digital Read Out (DRO) scales. It provides functionality similar to a normal machinist's DRO display, allowing the user to use the DRO scales on the machine when operating in a manual-only (hand-cranked) mode. It is most useful for manual machines such as DRO equipped Bridgeport style mills that have been converted to CNC but still have the manual controls.

MDRO is mouse and touch screen friendly.



Figure 5.35: MDRO Window

5.8.2 Getting Started

If your configuration is not currently set up to use MDRO, you can change it by editing the .ini file. In the [DISPLAY] section, change the DISPLAY = line to read DISPLAY = mdro. MDRO defaults to XYZ for the axes but that can be changed. Set [DISPLAY] section GEOMETRY = XYZ for a 3 axis mill. A lathe with DRO scales on the X and Z axes might use GEOMETRY = XZ.

When MDRO starts, a window like the one in the figure Figure 5.35 above opens.

5.8.2.1 INI File Options

Other options that can be included on the [DISPLAY] section include:

- MDRO_VAR_FILE = <file.var> - preload G54 - G57 coordinate system data.
 - Preload a .var file. This is typically the .var file used by the operational code.
- POINT_SIZE = <n> - Set text point size.

- This option sets the size of the font used which sets the overall size of the window. The default point size is 20, Typical sizes are 20 to 30.
- MM = 1 Set this if the DRO scales provide data scaled in millimeters.

5.8.2.2 Command Line Options

MDRO can be started by a `loadusr` command in a HAL file. Options equivalent to those in the INI file can be set on the command line:

- `-l <file.var>` - preload G54 - G57 coordinate system data.
- `-p <n>` - Set text point size.
- `-m` - Set this if the DRO scales provide data scaled in millimeters.
- `<axes>` - axes to display. See GEOMETRY above.

5.8.2.3 Pins

Using an example of "XYZA" for an AXES argument, these pins will be created when MDRO starts:

```
mdro.axis.0
mdro.axis.1
mdro.axis.2
mdro.axis.3
mdro.index-enable.0
mdro.index-enable.1
mdro.index-enable.2
mdro.index-enable.3
```

In this example, the first row of the display will be labeled X and will show the data from the DRO scale connected to pin `mdro.axis.0`. The `mdro.index-enable.n` pins should be connected to the index pins of the DRO if the DRO supports them.

The pins must be connected in the file specified in the `POSTGUI_HALFILE` entry of the `.ini` file when the program is started from a `.ini` file. They can be set directly after the `loadusr` command if the program is started in a HAL file.

5.8.3 MDRO Window

The MDRO window contains the following elements:

- A row for each axis. Each row includes:
 - The name of the axis
 - The current value
 - A "z" button that zeros the value
 - A "1/2" button that halves the value
 - A entry field that can be used to set a user-defined value. This field can be set from the keyboard or from the on-screen keypad.
 - A "I" button that starts an index operation (see below)
- A keypad used to set values in the entry field via a mouse or touchscreen.

- Coordinate system selection buttons
 - The "mcs" button selects the machine coordinate system. These are the raw values from the encoders connected to the `mdro.axis.n` pins.
 - The "cs1" - "cs4" buttons allow the user to select among one of four user-defined coordinate systems. If the program is started with the `MDRO_VAR_FILE =` option, the labels will be changed to "g54" - "g57" and the values from the specified `.var` file will be preloaded. Note that any changes to the values are not persistent: the `.var` file is never changed.
- Inch/Millimeter selection buttons.

5.8.4 Index operations

MDRO supports DRO scales with index marks. Hit the "I" button on the axis row then crank the axis to the index position. The machine coordinate will be zeroed. This is easiest to see at startup or when the "mcs" coordinate system has been selected.

5.8.5 Simulation

The easiest way to see how MDRO works is to try it in a simulation environment. Add this section to the end of your simulation HAL file, usually "hallib/core_sim.hal":

```
loadusr -W mdro -l sim.var XYZ
net x-pos-fb => mdro.axis.0
net y-pos-fb => mdro.axis.1
net z-pos-fb => mdro.axis.2
```

Chapter 6

Programming

6.1 Coordinate Systems

6.1.1 Introduction

In this chapter, we will try to demystify coordinate systems. It is a very important concept to understand the operation of a CNC machine, its configuration and its use.

We will also show that it is very interesting to use a reference point on the blank or the part and to make the program work from this point, without having to take into account where the part is placed on the table.

This chapter introduces you to offsets as they are used by the LinuxCNC. These include:

- Machine Coordinates (G53)
- Nine Coordinate System Offsets (G54-G59.3)
- Global Offsets (G92) and Local Offsets (G52)

6.1.2 Machine Coordinate System

When LinuxCNC is started the positions of each axis is the machine origin. Once an axis is homed, the machine origin for that axis is set to the homed position. The machine origin is the machine coordinate system on which all other coordinate systems are based. The [G53](#) G-code can be used to move in the machine coordinate system.

6.1.2.1 Machine coordinates moves: G53

Regardless of any offset that may be active, a G53 in a line of code tells the interpreter to move to the actual axes positions (absolute positions) specified. For example:

```
G53 G0 X0 Y0 Z0
```

will move from the current position to the position where the machine coordinates of the three axes will be at zero. You can use this command if you have a fixed position for the tool change or if your machine has an automatic tool changer. You can also use this command to clear the work area and access the workpiece in the vise.

G53 is a non modal command. It must be used in every block where a move in machine coordinate system is desired.

6.1.3 Coordinate Systems

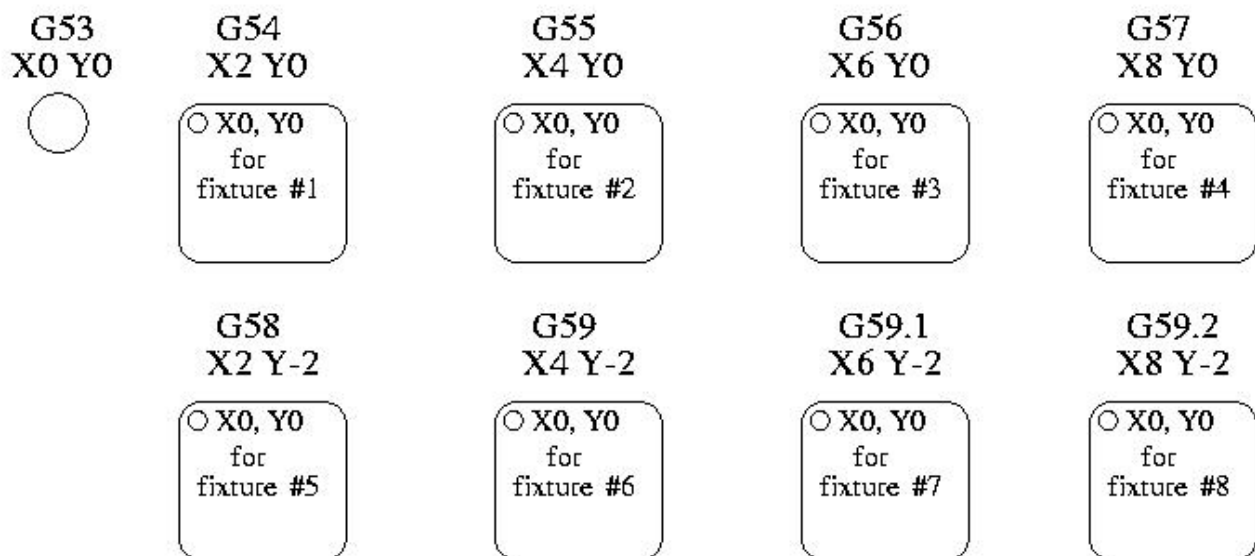


Figure 6.1: Coordinate Systems Example

Coordinate System Offsets

- G54 - use coordinate system 1
- G55 - use coordinate system 2
- G56 - use coordinate system 3
- G57 - use coordinate system 4
- G58 - use coordinate system 5
- G59 - use coordinate system 6
- G59.1 - use coordinate system 7
- G59.2 - use coordinate system 8
- G59.3 - use coordinate system 9

Coordinate system offsets are used to shift the coordinate system from the machine coordinate system. This allows the G-code to be programmed for the part without regard to the part location on the machine. Using coordinate system offsets would allow you to machine parts in multiple locations with the same G-code.

The values for offsets are stored in the VAR file that is requested by the INI file during the startup of an LinuxCNC. In the example below, which uses G55, the position of each axis for G55 origin is stored in a numbered variable.

In the VAR file scheme, the first variable number stores the X offset, the second the Y offset and so on for all nine axes. There are numbered sets like this for each of the coordinate system offsets.

Each of the graphical interfaces has a way to set values for these offsets. You can also set these values by editing the VAR file itself and then restart LinuxCNC so that the LinuxCNC reads the new values however this is not the recommended way. Using G10, G52, G92, G28.1, etc are better ways to set the variables. For our example, we will directly edit the file so that G55 will take the following values:

Table 6.1: Example of G55 parameters

Axis	Variable	Value
X	5241	2.000000
Y	5242	1.000000
Z	5243	-2.000000
A	5244	0.000000
B	5245	0.000000
C	5246	0.000000
U	5247	0.000000
V	5248	0.000000
W	5249	0.000000

You should read this as moving the zero positions of G55 to X = 2 units, Y= 1 unit, and Z = -2 units away from the absolute zero position.

Once there are values assigned, a call to G55 in a program block would shift the zero reference by the values stored. The following line would then move each axis to the new zero position. Unlike G53, G54 through G59.3 are modal commands. They will act on all blocks of code after one of them has been set. The program that might be run using fixture offsets would require only a single coordinate reference for each of the locations and all of the work to be done there. The following code is offered as an example of making a square using the G55 offsets that we set above.

```
G55 ; use coordinate system 2
G0 X0 Y0 Z0
G1 F2 Z-0.2000
X1
Y1
X0
Y0
G0 Z0
G54 ; use coordinate system 1
G0 X0 Y0 Z0
M2
```

In this example the G54 near the end leaves the G54 coordinate system with all zero offsets so that there is a modal code for the absolute machine based axis positions. This program assumes that we have done that and use the ending command as a command to machine zero. It would have been possible to use G53 and arrive at the same place but that command would not have been modal and any commands issued after it would have returned to using the G55 offsets because that coordinate system would still be in effect.

```
G54    uses parameters of coordinate system 1(((G54)))
G55    uses parameters of coordinate system 2(((G55)))
G56    uses parameters of coordinate system 3(((G56)))
G57    uses parameters of coordinate system 4(((G57)))
G58    uses parameters of coordinate system 5(((G58)))
G59    uses parameters of coordinate system 6(((G59)))
G59.1  uses parameters of coordinate system 7(((G59.1)))
G59.2  uses parameters of coordinate system 8(((G59.2)))
G59.3  uses parameters of coordinate system 9(((G59.3)))
```

6.1.3.1 Default Coordinate System

One other variable in the VAR file becomes important when we think about offset systems. This variable is named 5220. In the default files its value is set to 1.00000. This means that when the

LinuxCNC starts up it should use the first coordinate system as its default. If you set this to 9.00000 it would use the ninth offset system as its default for start up and reset. Any value other than an integer (decimal really) between 1 and 9, or a missing 5220 variable will cause the LinuxCNC to revert to the default value of 1.00000 on start up.

6.1.3.2 Setting Coordinate System Offsets

The G10 L2x command can be used to set coordinate system offsets:

- *G10 L2 P(1-9)* - Set offset(s) to a value. Current position irrelevant (see [G10 L2](#) for details).
- *G10 L20 P(1-9)* - Set offset(s) so current position becomes a value (see [G10 L20](#) for details).

Note

We only give a brief overview here, refer to the G-code sections for a full description.

6.1.4 Local and Global Offsets

6.1.4.1 The G52 command

G52 is used in a part program as a temporary "local coordinate system offset" within the workpiece coordinate system. An example use case is when machining several identical features at different locations on a part. For each feature, G52 programs a local reference point within workpiece coordinates, and a subprogram is called to machine the feature relative to that point.

G52 axis offsets are programmed relative to workpiece coordinate offsets G54 through G59.3. As a local offset, G52 is applied after the workpiece offset, including rotation. Thus, a part feature will be machined identically on each part regardless of the part's orientation on the pallet.

**Caution**

As a temporary offset, set and unset within the localized scope of a part program, in other G-code interpreters G52 does not persist after machine reset, M02 or M30. In LinuxCNC, G52 shares parameters with G92, which, for historical reasons, **does** persist these parameters. See [G92 Persistence Cautions](#) below.

**Caution**

G52 and G92 share the same offset registers. Therefore, setting G52 will override any earlier G92 setting, and G52 will persist across machine reset when G92 persistence is enabled. These interactions may result in unexpected offsets. See [G92 and G52 Interaction Cautions](#) below.

Programming G52 X1 Y2 offsets the current workpiece coordinate system X axis by 1 and Y axis by 2. Accordingly, on the DRO, the current tool position's X and Y coordinates will be reduced by 1 and 2, respectively. Axes unset in the command, such as Z in the previous example, will be unaffected: any previous G52 Z offset will remain in effect, and otherwise the Z offset will be zero.

The temporary local offset may be canceled with G52 X0 Y0. Any axes not explicitly zeroed will retain the previous offset.

G52 shares the same offset registers as G92, and thus G52 is visible on the DRO and preview labeled with G92.

6.1.5 G92 Axes Offsets

G92 is the most misunderstood and cleverest command programmable with LinuxCNC. The way it works has changed a bit between the first versions and the current one. These changes have doubt baffled many users. They should be seen as a command producing a temporary offset, which applies to all the other offsets.

6.1.5.1 The G92 commands

G92 is typically used in two conceptually different ways: as a "global coordinate system offset" or as a "local coordinate system offset".

The G92 set of commands includes:

- G92 - This command, when used with axis names, sets values to offset variables.
- G92.1 - This command sets zero values to the G92 variables.
- G92.2 - This command suspends but does not zero out the G92 variables.
- G92.3 - This command applies offset values that have been suspended.

As a global offset, G92 is used to shift all workpiece coordinate systems G54 through G59.3. An example use case is when machining several identical parts in fixtures with known locations on a pallet, but the pallet location may change between runs or between machines. Each fixture location offset, relative to a reference point on the pallet, is preset in one of the workpiece coordinate systems, G54 through G59.3, and G92 is used to "touch off" on the pallet reference point. Then, for each part, the corresponding workpiece coordinate system is selected and the part program is executed.

Note

G10 R- workpiece coordinate system rotation is specific to the *rs274ngc* interpreter, and the G92 offset is applied *after* rotation. When using G92 as a global offset, workpiece coordinate system rotations may have unexpected results.

As a local coordinate system, G92 is used as a temporary offset within the workpiece coordinate system. An example use case is when machining a part with several identical features at different locations. For each feature, G92 is used to set a local reference point, and a subprogram is called to machine the feature starting at that point.

Note

The use of G92 is discouraged for programming with local coordinate systems in a part program. Instead, see [G52](#), a local coordinate system offset more intuitive when desired offset relative to the workpiece is known but current tool location may not be known.

Programming G92 X0 Y0 Z0 sets the current tool location to the coordinates X0, Y0, and Z0, without motion. G92 **does not** work from absolute machine coordinates. It works from **current location**.

G92 also works from current location as modified by any other offsets that are in effect when the G92 command is invoked. While testing for differences between work offsets and actual offsets it was found that a G54 offset could cancel out a G92 and thus give the appearance that no offsets were in effect. However, the G92 was still in effect for all coordinates and did produce expected work offsets for the other coordinate systems.

By default, G92 offsets are restored after the machine is started. Programmers that wish for Fanuc behavior, where G92 offsets are cleared at machine start and after a reset or program end, may disable G92 persistence by setting `DISABLE_G92_PERSISTENCE = 1` in the `[RS274NGC]` section of the INI file.

Note

It is good practice to clear the G92 offsets at the end of their use with G92.1 or G92.2. When starting up LinuxCNC with G92 persistence enabled (the default), any offsets in the G92 variables will be applied when an axis is homed. See [G92 Persistence Cautions](#) below.

6.1.5.2 Setting G92 Values

There are at least two ways to set G92 values:

- With a right click on the position displays in tklinuxcnc, a window opens where it is possible to enter a value.
- With the G92 command

Both work from the current position of the axis that should be moved.

Programming *G92 X Y Z A B C U V W* sets the values of the G92 variables so that each axis takes the value associated with its name. Those values are assigned to the current position of the axes. These results satisfy paragraphs one and two of the NIST document.

G92 commands work from current axis location and add and subtract correctly to give the current axis position the value assigned by the G92 command. The effects work even though previous offsets are in.

So if the X axis is currently showing 2.0000 as its position a *G92 X0* will set an offset of -2.0000 so that the current location of X becomes zero. A *G92 X2* will set an offset of 0.0000 and the displayed position will not change. A *G92 X5.0000* will set an offset of 3.0000 so that the current displayed position becomes 5.0000.

6.1.5.3 G92 Persistence Cautions

By default, the values of a G92 offset will be saved in the VAR file and be restored after a machine reset or startup.

The G92 parameters are:

- 5210 - Enable/disable flag (1.0/0.0)
 - 5211 - X Axis Offset
 - 5212 - Y Axis Offset
 - 5213 - Z Axis Offset
 - 5214 - A Axis Offset
 - 5215 - B Axis Offset
 - 5216 - C Axis Offset
 - 5217 - U Axis Offset
 - 5218 - V Axis Offset
 - 5219 - W Axis Offset
-

where 5210 is the G92 enable flag (1 for enabled, 0 for disabled) and 5211 to 5219 are the axis offsets. If you are seeing unexpected positions as the result of a commanded move, as a result of storing an offset in a previous program and not clearing them at the end then issue a G92.1 in the MDI window to clear the stored offsets.

If G92 values exist in the VAR file when LinuxCNC starts up, the G92 values in the var file will be applied to the values of the current location of each axis. If this is home position and home position is set as machine zero everything will be correct. Once home has been established using real machine switches, or by moving each axis to a known home position and issuing an axis home command, any G92 offsets will be applied. If you have a G92 X1 in effect when you home the X axis the DRO will read X: 1.000 instead of the expected X: 0.000 because the G92 was applied to the machine origin. If you issue a G92.1 and the DRO now reads all zeros then you had a G92 offset in effect when you last ran LinuxCNC.

Unless your intention is to use the same G92 offsets in the next program, the best practice is to issue a G92.1 at the end of any G code files where you use G92 offsets.

When a program is aborted during processing that has G92 offsets in effect a startup will cause them to become active again. As a safeguard, always have your preamble to set the environment as you expect it. Additionally, G92 persistence may be disabled by setting `DISABLE_G92_PERSISTENCE = 1` in the `[RS274NGC]` section of the INI file.

6.1.5.4 G92 and G52 Interaction Cautions

G52 and G92 share the same offset registers. Unless G92 persistence is disabled in the INI file (see [G92 Commands](#)), G52 offsets will also persist after machine reset, M02 or M30. Beware that a G52 offset in effect during a program abort may result in unintended offsets when the next program is run. See [G92 Persistence Cautions](#) above.

6.1.6 Sample Programs Using Offsets

6.1.6.1 Sample Program Using Workpiece Coordinate Offsets

This sample engraving project mills a set of four .1 radius circles in roughly a star shape around a center circle. We can setup the individual circle pattern like this.

```
G10 L2 P1 X0 Y0 Z0 (ensure that G54 is set to machine zero)
G0 X-0.1 Y0 Z0
G1 F1 Z-0.25
G3 X-0.1 Y0 I0.1 J0
G0 Z0
M2
```

We can issue a set of commands to create offsets for the four other circles like this.

```
G10 L2 P2 X0.5 (offsets G55 X value by 0.5 inch)
G10 L2 P3 X-0.5 (offsets G56 X value by -0.5 inch)
G10 L2 P4 Y0.5 (offsets G57 Y value by 0.5 inch)
G10 L2 P5 Y-0.5 (offsets G58 Y value by -0.5 inch)
```

We put these together in the following program:

(a program for milling five small circles in a diamond shape)

```
G10 L2 P1 X0 Y0 Z0 (ensure that G54 is machine zero)
G10 L2 P2 X0.5 (offsets G55 X value by 0.5 inch)
G10 L2 P3 X-0.5 (offsets G56 X value by -0.5 inch)
G10 L2 P4 Y0.5 (offsets G57 Y value by 0.5 inch)
```

```
G10 L2 P5 Y-0.5 (offsets G58 Y value by -0.5 inch)

G54 G0 X-0.1 Y0 Z0 (center circle)
G1 F1 Z-0.25
G3 X-0.1 Y0 I0.1 J0
G0 Z0

G55 G0 X-0.1 Y0 Z0 (first offset circle)
G1 F1 Z-0.25
G3 X-0.1 Y0 I0.1 J0
G0 Z0

G56 G0 X-0.1 Y0 Z0 (second offset circle)
G1 F1 Z-0.25
G3 X-0.1 Y0 I0.1 J0
G0 Z0

G57 G0 X-0.1 Y0 Z0 (third offset circle)
G1 F1 Z-0.25
G3 X-0.1 Y0 I0.1 J0
G0 Z0

G58 G0 X-0.1 Y0 Z0 (fourth offset circle)
G1 F1 Z-0.25
G3 X-0.1 Y0 I0.1 J0
G54 G0 X0 Y0 Z0

M2
```

Now comes the time when we might apply a set of G92 offsets to this program. You'll see that it is running in each case at Z0. If the mill were at the zero position, a G92 Z1.0000 issued at the head of the program would shift everything an inch. You might also shift the whole pattern around in the XY plane by adding some X and Y offsets with G92. If you do this you should add a G92.1 command just before the M2 that ends the program. If you do not, other programs that you might run after this one will also use that G92 offset. Furthermore it would save the G92 values when you shut down the LinuxCNC and they will be recalled when you start up again.

6.1.6.2 Sample Program Using G52 Offsets

(To be written)

6.2 G-Code Overview

6.2.1 Overview

The LinuxCNC G-code language is based on the RS274/NGC language. The G Code language is based on lines of code. Each line (also called a *block*) may include commands to do several different things. Lines of code may be collected in a file to make a program.

A typical line of code consists of an optional line number at the beginning followed by one or more *words*. A word consists of a letter followed by a number (or something that evaluates to a number). A word may either give a command or provide an argument to a command. For example, *G1 X3* is a valid line of code with two words. *G1* is a command meaning *move in a straight line at the programmed feed rate to the programmed end point*, and *X3* provides an argument value (the value of X should be 3 at the end of the move). Most LinuxCNC G-code commands start with either G or M (for General and Miscellaneous). The words for these commands are called *G codes* and *M-codes*.

The LinuxCNC language has no indicator for the start of a program. The Interpreter, however, deals with files. A single program may be in a single file, or a program may be spread across several files. A file may be demarcated with percents in the following way. The first non-blank line of a file may contain nothing but a percent sign, %, possibly surrounded by white space, and later in the file (normally at the end of the file) there may be a similar line. Demarcating a file with percents is optional if the file has an *M2* or *M30* in it, but is required if not. An error will be signaled if a file has a percent line at the beginning but not at the end. The useful contents of a file demarcated by percents stop after the second percent line. Anything after that is ignored.

The LinuxCNC G-code language has two commands (*M2* or *M30*), either of which ends a program. A program may end before the end of a file. Lines of a file that occur after the end of a program are not to be executed. The interpreter does not even read them.

6.2.2 Format of a line

A permissible line of input code consists of the following, in order, with the restriction that there is a maximum (currently 256) to the number of characters allowed on a line.

- an optional block delete character, which is a slash /.
- an optional line number.
- any number of words, parameter settings, and comments.
- an end of line marker (carriage return or line feed or both).

Any input not explicitly allowed is illegal and will cause the Interpreter to signal an error.

Spaces and tabs are allowed anywhere on a line of code and do not change the meaning of the line, except inside comments. This makes some strange-looking input legal. The line *G0X +0. 12 34Y 7* is equivalent to *G0 x+0.1234 Y7*, for example.

Blank lines are allowed in the input. They are to be ignored.

Input is case insensitive, except in comments, i.e., any letter outside a comment may be in upper or lower case without changing the meaning of a line.

6.2.2.1 /: Block Delete

The optional block delete character the slash / when placed first on a line can be used by some user interfaces to skip lines of code when needed. In Axis the key combination Alt-m-/ toggles block delete on and off. When block delete is on any lines starting with the slash / are skipped.

In AXIS, it is also possible to enable block delete with the following icon:

AXIS Block Delete Icon 

6.2.2.2 Line Number

A line number is the letter N followed by an unsigned integer, optionally followed by a period and another unsigned integer. For example, *N1234* and *N56.78* are valid line numbers. They may be repeated or used out of order, although normal practice is to avoid such usage. Line numbers may also be skipped, and that is normal practice. A line number is not required to be used, but must be in the proper place if used.

6.2.2.3 Word

A word is a letter other than N followed by a real value.

Words may begin with any of the letters shown in the following Table. The table includes N for completeness, even though, as defined above, line numbers are not words. Several letters (I, J, K, L, P, R) may have different meanings in different contexts. Letters which refer to axis names are not valid on a machine which does not have the corresponding axis.

Table 6.2: Words and their meanings

Letter	Meaning
A	A axis of machine
B	B axis of machine
C	C axis of machine
D	Tool radius compensation number
F	Feed rate
G	General function (See table Modal Groups)
H	Tool length offset index
I	X offset for arcs and G87 canned cycles
J	Y offset for arcs and G87 canned cycles
K	Z offset for arcs and G87 canned cycles.
	Spindle-Motion Ratio for G33 synchronized movements.
L	generic parameter word for G10, M66 and others
M	Miscellaneous function (See table Modal Groups)
N	Line number
P	Dwell time in canned cycles and with G4.
	Key used with G10.
Q	Feed increment in G73, G83 canned cycles
R	Arc radius or canned cycle plane
S	Spindle speed
T	Tool selection
U	U axis of machine
V	V axis of machine
W	W axis of machine
X	X axis of machine
Y	Y axis of machine
Z	Z axis of machine

6.2.2.4 Numbers

The following rules are used for (explicit) numbers. In these rules a digit is a single character between 0 and 9.

- A number consists of:
 - an optional plus or minus sign, followed by
 - zero to many digits, followed, possibly, by
 - one decimal point, followed by
 - zero to many digits - provided that there is at least one digit somewhere in the number.
- There are two kinds of numbers:

- Integers, that does not have a decimal point,
- Decimals, that do have a decimal point.
- Numbers may have any number of digits, subject to the limitation on line length. Only about seventeen significant figures will be retained, however (enough for all known applications).
- A non-zero number with no sign but the first character is assumed to be positive.

Notice that initial (before the decimal point and the first non-zero digit) and trailing (after the decimal point and the last non-zero digit) zeros are allowed but not required. A number written with initial or trailing zeros will have the same value when it is read as if the extra zeros were not there.

Numbers used for specific purposes in RS274/NGC are often restricted to some finite set of values or some to some range of values. In many uses, decimal numbers must be close to integers; this includes the values of indices (for parameters and carousel slot numbers, for example), M-codes, and G-codes multiplied by ten. A decimal number which is intended to represent an integer is considered close enough if it is within 0.0001 of an integer value.

6.2.3 Parameters

The RS274/NGC language supports *parameters* - what in other programming languages would be called *variables*. There are several types of parameter of different purpose and appearance, each described in the following sections. The only value type supported by parameters is floating-point; there are no string, boolean or integer types in G-code like in other programming languages. However, logic expressions can be formulated with [boolean operators](#) (*AND*, *OR*, *XOR*, and the comparison operators *EQ*, *NE*, *GT*, *GE*, *LT*, *LE*), and the *MOD*, *ROUND*, *FUP* and *FIX* [operators](#) support integer arithmetic.

Parameters differ in syntax, scope, behavior when not yet initialized, mode, persistence and intended use.

Syntax

There are three kinds of syntactic appearance:

- *numbered* - #4711
- *named local* - #<localvalue>
- *named global* - #<_globalvalue>

Scope

The scope of a parameter is either global, or local within a subroutine. Subroutine parameters and local named variables have local scope. Global named parameters and numbered parameters starting from number 31 are global in scope. RS274/NGC uses *lexical scoping* - in a subroutine only the local variables defined therein, and any global variables are visible. The local variables of a calling procedure are not visible in a called procedure.

Behavior of uninitialized parameters

- Uninitialized global parameters, and unused subroutine parameters return the value zero when used in an expression.
- Uninitialized named parameters signal an error when used in an expression.

Mode

Most parameters are read/write and may be assigned to within an assignment statement. However, for many predefined parameters this does not make sense, so they are read-only - they may appear in expressions, but not on the left-hand side of an assignment statement.

Persistence

When LinuxCNC is shut down, volatile parameters lose their values. All parameters except numbered parameters in the current persistent range ¹ are volatile. Persistent parameters are saved in the .var file and restored to their previous values when LinuxCNC is started again. Volatile numbered parameters are reset to zero.

Intended Use

- user parameters - numbered parameters in the range 31..5000, and named global and local parameters except predefined parameters. These are available for general-purpose storage of floating-point values, like intermediate results, flags etc, throughout program execution. They are read/write (can be assigned a value).
- [subroutine parameters](#) - these are used to hold the actual parameters passed to a subroutine.
- [numbered parameters](#) - most of these are used to access offsets of coordinate systems.
- [system parameters](#) - used to determine the current running version. They are read-only.

6.2.3.1 Numbered Parameters

A numbered parameter is the pound character # followed by an integer between 1 and (currently) 5602 ². The parameter is referred to by this integer, and its value is whatever number is stored in the parameter.

A value is stored in a parameter with the = operator; for example:

```
#3 = 15 (set parameter 3 to 15)
```

A parameter setting does not take effect until after all parameter values on the same line have been found. For example, if parameter 3 has been previously set to 15 and the line `#3=6 G1 X#3` is interpreted, a straight move to a point where X equals 15 will occur and the value of parameter 3 will be 6.

The # character takes precedence over other operations, so that, for example, `#1+2` means the number found by adding 2 to the value of parameter 1, not the value found in parameter 3. Of course, `#[1+2]` does mean the value found in parameter 3. The # character may be repeated; for example `##2` means the value of the parameter whose index is the (integer) value of parameter 2.

- 31-5000 - G-code user parameters. These parameters are global in the G code file, and available for general use. Volatile.
- 5061-5069 - Coordinates of a [G38](#) probe result (X, Y, Z, A, B, C, U, V & W). Coordinates are in the coordinate system in which the G38 took place. Volatile.
- 5070 - [G38](#) probe result: 1 if success, 0 if probe failed to close. Used with G38.3 and G38.5. Volatile.
- 5161-5169 - "G28" Home for X, Y, Z, A, B, C, U, V & W. Persistent.
- 5181-5189 - "G30" Home for X, Y, Z, A, B, C, U, V & W. Persistent.
- 5210 - 1 if "G52" or "G92" offset is currently applied, 0 otherwise. Volatile by default; persistent if `DISABLE_G92_PERSISTENCE = 1` in the `[RS274NGC]` section of the INI file.
- 5211-5219 - Shared "G52" and "G92" offset for X, Y, Z, A, B, C, U, V & W. Volatile by default; persistent if `DISABLE_G92_PERSISTENCE = 1` in the `[RS274NGC]` section of the INI file.
- 5220 - Coordinate System number 1 - 9 for G54 - G59.3. Persistent.

¹ The range of persistent parameters may change as development progresses. This range is currently 5161- 5390. It is defined in the `_required_parameters` array in file `src/emc/rs274ngc/interp_array.cc`.

²The RS274/NGC interpreter maintains an array of numbered parameters. Its size is defined by the symbol `RS274NGC_MAX_PARAMETERS` in the file `src/emc/rs274ngc/interp_internal.hh`). This number of numerical parameters may also increase as development adds support for new parameters.

- 5221-5230 - Coordinate System 1, G54 for X, Y, Z, A, B, C, U, V, W & R. R denotes the XY rotation angle around the Z axis. Persistent.
- 5241-5250 - Coordinate System 2, G55 for X, Y, Z, A, B, C, U, V, W & R. Persistent.
- 5261-5270 - Coordinate System 3, G56 for X, Y, Z, A, B, C, U, V, W & R. Persistent.
- 5281-5290 - Coordinate System 4, G57 for X, Y, Z, A, B, C, U, V, W & R. Persistent.
- 5301-5310 - Coordinate System 5, G58 for X, Y, Z, A, B, C, U, V, W & R. Persistent.
- 5321-5330 - Coordinate System 6, G59 for X, Y, Z, A, B, C, U, V, W & R. Persistent.
- 5341-5350 - Coordinate System 7, G59.1 for X, Y, Z, A, B, C, U, V, W & R. Persistent.
- 5361-5370 - Coordinate System 8, G59.2 for X, Y, Z, A, B, C, U, V, W & R. Persistent.
- 5381-5390 - Coordinate System 9, G59.3 for X, Y, Z, A, B, C, U, V, W & R. Persistent.
- 5399 - Result of M66 - Check or wait for input. Volatile.
- 5400 - Tool Number. Volatile.
- 5401-5409 - Tool Offsets for X, Y, Z, A, B, C, U, V & W. Volatile.
- 5410 - Tool Diameter. Volatile.
- 5411 - Tool Front Angle. Volatile.
- 5412 - Tool Back Angle. Volatile.
- 5413 - Tool Orientation. Volatile.
- 5420-5428 - Current relative position in the active coordinate system including all offsets and in the current program units for X, Y, Z, A, B, C, U, V & W, volatile.
- 5599 - Flag for controlling the output of (DEBUG,) statements. 1=output, 0=no output; default=1. Volatile.
- 5600 - Toolchanger fault indicator. Used with the iocontrol-v2 component. 1: toolchanger faulted, 0: normal. Volatile.
- 5601 - Toolchanger fault code. Used with the iocontrol-v2 component. Reflects the value of the *toolchanger-reason* HAL pin if a fault occurred. Volatile.

Numbered Parameters Persistence The values of parameters in the persistent range are retained over time, even if the machining center is powered down. LinuxCNC uses a parameter file to ensure persistence. It is managed by the Interpreter. The Interpreter reads the file when it starts up, and writes the file when it exits.

The format of a parameter file is shown in Table [Parameter File Format](#).

The Interpreter expects the file to have two columns. It skips any lines which do not contain exactly two numeric values. The first column is expected to contain an integer value (the parameter's number). The second column contains a floating point number (this parameter's last value). The value is represented as a double-precision floating point number inside the Interpreter, but a decimal point is not required in the file.

Parameters in the user-defined range (31-5000) may be added to this file. Such parameters will be read by the Interpreter and written to the file as it exits.

Missing Parameters in the persistent range will be initialized to zero and written with their current values on the next save operation.

The parameter numbers must be arranged in ascending order. An *Parameter file out of order* error will be signaled if they are not in ascending order.

The original file is saved as a backup file when the new file is written.

Table 6.3: Parameter File Format

Parameter Number	Parameter Value
5161	0.0
5162	0.0

6.2.3.2 Subroutine Parameters

- 1-30 Subroutine local parameters of call arguments. These parameters are local to the subroutine. Volatile. See also the chapter on [O-Codes](#).

6.2.3.3 Named Parameters

Named parameters work like numbered parameters but are easier to read. All parameter names are converted to lower case and have spaces and tabs removed, so `<param>` and `<P a R a m >` refer to the same parameter. Named parameters must be enclosed with `< >` marks.

`#<named parameter>` is a local named parameter. By default, a named parameter is local to the scope in which it is assigned. You can't access a local parameter outside of its subroutine. This means that two subroutines can use the same parameter names without fear of one subroutine overwriting the values in another.

`#<_global named parameter>` is a global named parameter. They are accessible from within called subroutines and may set values within subroutines that are accessible to the caller. As far as scope is concerned, they act just like regular numeric parameters. They are not stored in files.

Examples:

Declaration of named global variable

```
#<_endmill_dia> = 0.049
```

Reference to previously declared global variable

```
#<_endmill_rad> = [#<_endmill_dia>/2.0]
```

Mixed literal and named parameters

```
o100 call [0.0] [0.0] [#<_inside_cutout>-#<_endmill_dia>] [#<_Zcut>] [#<_feedrate>]
```

Named parameters spring into existence when they are assigned a value for the first time. Local named parameters vanish when their scope is left: when a subroutine returns, all its local parameters are deleted and cannot be referred to anymore.

It is an error to use a non-existent named parameter within an expression, or at the right-hand side of an assignment. Printing the value of a non-existent named parameter with a `DEBUG` statement - like (`DEBUG, <no_such_parameter>`) will display the string `#`.

Global parameters, as well as local parameters assigned to at the global level, retain their value once assigned even when the program ends, and have these values when the program is run again.

The [EXISTS function](#) tests whether a given named parameter exists.

6.2.3.4 Predefined Named Parameters

The following global read only named parameters are available to access internal state of the interpreter and machine state. They can be used in arbitrary expressions, for instance to control flow of the program with if-then-else statements. Note that new [predefined named parameters](#) can be added easily without changes to the source code.

- `#<_vmajor>` - Major package version. If current version was 2.5.2 would return 2.5.
- `#<_vminor>` - Minor package version. If current version was 2.6.2 it would return 0.2.
- `#<_line>` - Sequence number. If running a G-code file, this returns the current line number.
- `#<_motion_mode>` - Return the interpreter's current motion mode:

Motion mode	return value
G1	10
G2	20
G3	30
G33	330
G38.2	382
G38.3	383
G38.4	384
G38.5	385
G5.2	52
G73	730
G76	760
G80	800
G81	810
G82	820
G83	830
G84	840
G85	850
G86	860
G87	870
G88	880
G89	890

- `#<_plane>` - returns the value designating the current plane:

Plane	return value
G17	170
G18	180
G19	190
G17.1	171
G18.1	181
G19.1	191

- `#<_ccomp>` - Status of cutter compensation. Return values:

Mode	return value
G40	400
G41	410
G41.1	411
G41	410
G42	420
G42.1	421

- `#<_metric>` - Return 1 if G21 is on, else 0.
- `#<_imperial>` - Return 1 if G20 is on, else 0.
- `#<_absolute>` - Return 1 if G90 is on, else 0.
- `#<_incremental>` - Return 1 if G91 is on, else 0.
- `#<_inverse_time>` - Return 1 if inverse feed mode (G93) is on, else 0.
- `#<_units_per_minute>` - Return 1 if Units/minute feed mode (G94) is on, else 0.
- `#<_units_per_rev>` - Return 1 if Units/revolution mode (G95) is on, else 0.
- `#<_coord_system>` - Return a float of the current coordinate system name (G54..G59.3). For example if your in G55 coordinate system the return value is 550.000000 and if your in G59.1 the return value is 591.000000.

Mode	return value
G54	540
G55	550
G56	560
G57	570
G58	580
G59	590
G59.1	591
G59.2	592
G59.3	593

- `#<_tool_offset>` - Return 1 if tool offset (G43) is on, else 0.
- `#<_retract_r_plane>` - Return 1 if G98 is set, else 0.
- `#<_retract_old_z>` - Return 1 if G99 is on, else 0.

6.2.3.5 System Parameters

- `#<_spindle_rpm_mode>` - Return 1 if spindle rpm mode (G97) is on, else 0.
- `#<_spindle_css_mode>` - Return 1 if constant surface speed mode (G96) is on, else 0.
- `#<_ijk_absolute_mode>` - Return 1 if Absolute Arc distance mode (G90.1) is on, else 0.
- `#<_lathe_diameter_mode>` - Return 1 if this is a lathe configuration and diameter (G7) mode is on, else 0.
- `#<_lathe_radius_mode>` - Return 1 if this is a lathe configuration and radius (G8) mode is on, else 0.

- `#<_spindle_on>` - Return 1 if spindle currently running (M3 or M4) else 0.
 - `#<_spindle_cw>` - Return 1 if spindle direction is clockwise (M3) else 0.
 - `#<_mist>` - Return 1 if mist (M7) is on.
 - `#<_flood>` - Return 1 if flood (M8) is on.
 - `#<_speed_override>` - Return 1 if feed override (M48 or M50 P1) is on, else 0.
 - `#<_feed_override>` - Return 1 if feed override (M48 or M51 P1) is on, else 0.
 - `#<_adaptive_feed>` - Return 1 if adaptive feed (M52 or M52 P1) is on, else 0.
 - `#<_feed_hold>` - Return 1 if feed hold switch is enabled (M53 P1), else 0.
 - `#<_feed>` - Return the current value of F, not the actual feed rate.
 - `#<_rpm>` - Return the current value of S, not the actual spindle speed.
 - `#<_x>` - Return current relative X coordinate including all offsets. Same as #5420. In a lathe configuration, it always returns radius.
 - `#<_y>` - Return current relative Y coordinate including all offsets. Same as #5421.
 - `#<_z>` - Return current relative Z coordinate including all offsets. Same as #5422.
 - `#<_a>` - Return current relative A coordinate including all offsets. Same as #5423.
 - `#<_b>` - Return current relative B coordinate including all offsets. Same as #5424.
 - `#<_c>` - Return current relative C coordinate including all offsets. Same as #5425.
 - `#<_u>` - Return current relative U coordinate including all offsets. Same as #5426.
 - `#<_v>` - Return current relative V coordinate including all offsets. Same as #5427.
 - `#<_w>` - Return current relative W coordinate including all offsets. Same as #5428.
 - `#<_abs_x>` - Return current absolute X coordinate (G53) including no offsets.
 - `#<_abs_y>` - Return current absolute Y coordinate (G53) including no offsets.
 - `#<_abs_z>` - Return current absolute Z coordinate (G53) including no offsets.
 - `#<_abs_a>` - Return current absolute A coordinate (G53) including no offsets.
 - `#<_abs_b>` - Return current absolute B coordinate (G53) including no offsets.
 - `#<_abs_c>` - Return current absolute C coordinate (G53) including no offsets.
 - `#<_current_tool>` - Return number of the current tool in spindle. Same as #5400.
 - `#<_current_pocket>` - Return the tooldata index for the current tool.
 - `#<_selected_tool>` - Return number of the selected tool post a T code. Default -1.
 - `#<_selected_pocket>` - Return the tooldata index of the selected pocket post a T code. Default -1 (no pocket selected).
 - `#<_value>` - Return value from the last O-word *return* or *endsub*. Default value 0 if no expression after *return* or *endsub*. Initialized to 0 on program start.
 - `#<_value_returned>` - 1.0 if the last O-word *return* or *endsub* returned a value, 0 otherwise. Cleared by the next O-word call.
-

- `#<_task>` - 1.0 if the executing interpreter instance is part of milltask, 0.0 otherwise. Sometimes it is necessary to treat this case specially to retain proper preview, for instance when testing the success of a probe (G38.n) by inspecting #5070, which will always fail in the preview interpreter (e.g. Axis).
- `#<_call_level>` - current nesting level of O-word procedures. For debugging.
- `#<_remap_level>` - current level of the remap stack. Each remap in a block adds one to the remap level. For debugging.

6.2.4 HAL pins and INI values

If enabled in the [INI file](#) G-code has access to the values of INI file entries and HAL pins.

- `#<_ini[section]name>` Returns the value of the corresponding item in the INI file.

For example, if the INI file looks like so:

```
[SETUP]
XPOS = 3.145
YPOS = 2.718
```

you may refer to the named parameters `#<_ini[setup]xpos>` and `#<_ini[setup]ypos>` within G-code.

EXISTS can be used to test for presence of a given INI file variable:

```
o100 if [EXISTS[#<_ini[setup]xpos>]]
  (debug, [setup]xpos exists: #<_ini[setup]xpos>)
o100 else
  (debug, [setup]xpos does not exist)
o100 endif
```

The value is read from the INI file once, and cached in the interpreter. These parameters are read-only - assigning a value will cause a runtime error. The names are not case sensitive - they are converted to uppercase before consulting the INI file.

- `#<_hal[HAL item]>` Allows G-code programs to read the values of HAL pins. Variable access is read-only, the only way to *set* HAL pins from G-code remains M62-M65, M67, M68 and custom M100-M199 codes. Note that the value read will not update in real-time, typically the value that was on the pin when the G-code program was started will be returned. It is possible to work round this by forcing a state synch. One way to do this is with a dummy M66 command: M66E0L0

Example:

```
(debug, #<_hal[motion-controller.time]>)
```

Access of HAL items is read-only. Currently, only all-lowercase HAL names can be accessed this way.

EXISTS can be used to test for the presence of a given HAL item:

```
o100 if [EXISTS[#<_hal[motion-controller.time]>]]
  (debug, [motion-controller.time] exists: #<_hal[motion-controller.time]>)
o100 else
  (debug, [motion-controller.time] does not exist)
o100 endif
```

This feature was motivated by the desire for stronger coupling between user interface components like GladeVCP and PyVCP to act as parameter source for driving NGC file behavior. The alternative - going through the M6x pins and wiring them - has a limited, non-mnemonic namespace and is unnecessarily cumbersome just as a UI/Interpreter communications mechanism.

6.2.5 Expressions

An expression is a set of characters starting with a left bracket `[` and ending with a balancing right bracket `]`. In between the brackets are numbers, parameter values, mathematical operations, and other expressions. An expression is evaluated to produce a number. The expressions on a line are evaluated when the line is read, before anything on the line is executed. An example of an expression is `[1 + acos[0] - [#3 ** [4.0/2]]]`.

6.2.6 Binary Operators

Binary operators only appear inside expressions. There are four basic mathematical operations: addition (+), subtraction (-), multiplication (*), and division (/). There are three logical operations: non-exclusive or (OR), exclusive or (XOR), and logical and (AND). The eighth operation is the modulus operation (MOD). The ninth operation is the *power* operation (**) of raising the number on the left of the operation to the power on the right. The relational operators are equality (EQ), inequality (NE), strictly greater than (GT), greater than or equal to (GE), strictly less than (LT), and less than or equal to (LE).

The binary operations are divided into several groups according to their precedence. If operations in different precedence groups are strung together (for example in the expression `[2.0 / 3 * 1.5 - 5.5 / 11.0]`), operations in a higher group are to be performed before operations in a lower group. If an expression contains more than one operation from the same group (such as the first / and * in the example), the operation on the left is performed first. Thus, the example is equivalent to: `[[[2.0 / 3] * 1.5] - [5.5 / 11.0]]`, which is equivalent to `[1.0 - 0.5]`, which is 0.5.

The logical operations and modulus are to be performed on any real numbers, not just on integers. The number zero is equivalent to logical false, and any non-zero number is equivalent to logical true.

Table 6.4: Operators Precedence

Operators	Precedence
**	<i>highest</i>
* / MOD	
+ -	
EQ NE GT GE LT LE	
AND OR XOR	<i>lowest</i>

6.2.7 Equality and floating-point values

The RS274/NGC language only supports floating-point values of finite precision. Therefore, testing for equality or inequality of two floating-point values is inherently problematic. The interpreter solves this problem by considering values equal if their absolute difference is less than 0.0001 (this value is defined as `TOLERANCE_EQUAL` in `src/emc/rs274ngc/interp_internal.hh`).

6.2.8 Functions

The available functions are shown in following table. Arguments to unary operations which take angle measures (COS, SIN, and TAN) are in degrees. Values returned by unary operations which return angle measures (ACOS, ASIN, and ATAN) are also in degrees.

Table 6.5: G-code Functions

Function Name	Function result
ATAN[arg]/[arg]	Four quadrant inverse tangent
ABS[arg]	Absolute value
ACOS[arg]	Inverse cosine
ASIN[arg]	Inverse sine
COS[arg]	Cosine
EXP[arg]	e raised to the given power
FIX[arg]	Round down to integer
FUP[arg]	Round up to integer
ROUND[arg]	Round to nearest integer
LN[arg]	Base-e logarithm
SIN[arg]	Sine
SQRT[arg]	Square Root
TAN[arg]	Tangent
EXISTS[arg]	Check named Parameter

The *FIX* function rounds towards the left (less positive or more negative) on a number line, so that $FIX[2.8] = 2$ and $FIX[-2.8] = -3$.

The *FUP* operation rounds towards the right (more positive or less negative) on a number line; $FUP[2.8] = 3$ and $FUP[-2.8] = -2$.

The *EXISTS* function checks for the existence of a single named parameter. It takes only one named parameter and returns 1 if it exists and 0 if it does not exist. It is an error if you use a numbered parameter or an expression. Here is an example for the usage of the *EXISTS* function:

```
o<test> sub
o10 if [EXISTS[#<_global>]]
    (debug, _global exists and has the value #<_global>)
o10 else
    (debug, _global does not exist)
o10 endif
o<test> endsub

o<test> call
#<_global> = 4711
o<test> call
m2
```

6.2.9 Repeated Items

A line may have any number of G words, but two G words from the same modal group may not appear on the same line. See the [Modal Groups](#) section for more information.

A line may have zero to four M words. Two M words from the same modal group may not appear on the same line.

For all other legal letters, a line may have only one word beginning with that letter.

If a parameter setting of the same parameter is repeated on a line, $\#3=15 \#3=6$, for example, only the last setting will take effect. It is silly, but not illegal, to set the same parameter twice on the same line.

If more than one comment appears on a line, only the last one will be used; each of the other comments will be read and its format will be checked, but it will be ignored thereafter. It is expected that putting more than one comment on a line will be very rare.

6.2.10 Item order

The three types of item whose order may vary on a line (as given at the beginning of this section) are word, parameter setting, and comment. Imagine that these three types of item are divided into three groups by type.

The first group (the words) may be reordered in any way without changing the meaning of the line.

If the second group (the parameter settings) is reordered, there will be no change in the meaning of the line unless the same parameter is set more than once. In this case, only the last setting of the parameter will take effect. For example, after the line `#3=15 #3=6` has been interpreted, the value of parameter 3 will be 6. If the order is reversed to `#3=6 #3=15` and the line is interpreted, the value of parameter 3 will be 15.

If the third group (the comments) contains more than one comment and is reordered, only the last comment will be used.

If each group is kept in order or reordered without changing the meaning of the line, then the three groups may be interleaved in any way without changing the meaning of the line. For example, the line `g40 g1 #3=15 (foo) #4=-7.0` has five items and means exactly the same thing in any of the 120 possible orders (such as `#4=-7.0 g1 #3=15 g40 (foo)`) for the five items.

6.2.11 Commands and Machine Modes

Many commands cause the controller to change from one mode to another, and the mode stays active until some other command changes it implicitly or explicitly. Such commands are called *modal*. For example, if coolant is turned on, it stays on until it is explicitly turned off. The G-codes for motion are also modal. If a G1 (straight move) command is given on one line, for example, it will be executed again on the next line if one or more axis words is available on the line, unless an explicit command is given on that next line using the axis words or canceling motion.

Non-modal codes have effect only on the lines on which they occur. For example, G4 (dwell) is non-modal.

6.2.12 Polar Coordinates

Polar Coordinates can be used to specify the XY coordinate of a move. The `@n` is the distance and `^n` is the angle. The advantage of this is for things like bolt hole circles which can be done very simply by moving to a point in the center of the circle, setting the offset and then moving out to the first hole then run the drill cycle. Polar Coordinates always are from the current XY zero position. To shift the Polar Coordinates from machine zero use an offset or select a coordinate system.

In Absolute Mode the distance and angle is from the XY zero position and the angle starts with 0 on the X Positive axis and increases in a CCW direction about the Z axis. The code `G1 @1 ^90` is the same as `G1 Y1`.

In Relative Mode the distance and angle is also from the XY zero position but it is cumulative. This can be confusing at first how this works in incremental mode.

For example if you have the following program you might expect it to be a square pattern:

```
F100 G1 @.5 ^90
G91 @.5 ^90
@.5 ^90
@.5 ^90
@.5 ^90
G90 G0 X0 Y0 M2
```

You can see from the following figure that the output is not what you might expect. Because we added 0.5 to the distance each time the distance from the XY zero position increased with each line.

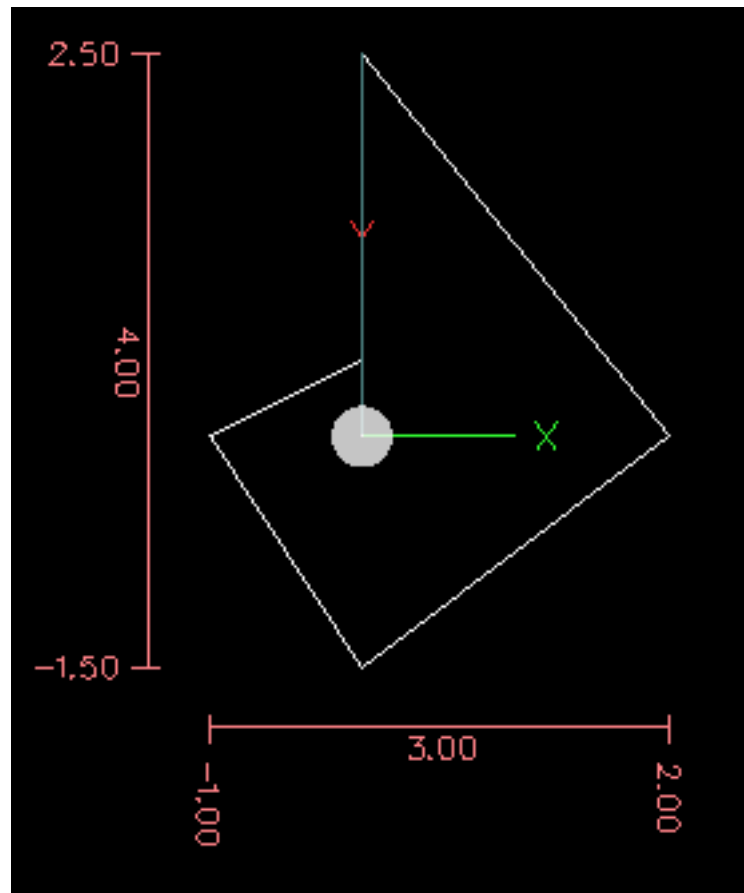


Figure 6.2: Polar Spiral

The following code will produce our square pattern:

```
F100 G1 @.5 ^90
G91 ^90
^90
^90
^90
G90 G0 X0 Y0 M2
```

As you can see by only adding to the angle by 90 degrees each time the end point distance is the same for each line.



Figure 6.3: Polar Square

It is an error if:

- An incremental move is started at the origin
- A mix of Polar and X or Y words are used

6.2.13 Modal Groups

Modal commands are arranged in sets called *modal groups*, and only one member of a modal group may be in force at any given time. In general, a modal group contains commands for which it is logically impossible for two members to be in effect at the same time - like measure in inches vs. measure in millimeters. A machining center may be in many modes at the same time, with one mode from each modal group being in effect. The modal groups are shown in the following Table.

Table 6.6: G-Code Modal Groups

Modal Group Meaning	Member Words
Non-modal codes (Group 0)	G4, G10 G28, G30, G52, G53, G92, G92.1, G92.2, G92.3,
Motion (Group 1)	G0, G1, G2, G3, G33, G38.n, G73, G76, G80, G81
	G82, G83, G84, G85, G86, G87, G88, G89
Plane selection (Group 2)	G17, G18, G19, G17.1, G18.1, G19.1
Distance Mode (Group 3)	G90, G91

Table 6.6: (continued)

Modal Group Meaning	Member Words
Arc IJK Distance Mode (Group 4)	G90.1, G91.1
Feed Rate Mode (Group 5)	G93, G94, G95
Units (Group 6)	G20, G21
Cutter Diameter Compensation (Group 7)	G40, G41, G42, G41.1, G42.1
Tool Length Offset (Group 8)	G43, G43.1, G49
Canned Cycles Return Mode (Group 10)	G98, G99
Coordinate System (Group 12)	G54, G55, G56, G57, G58, G59, G59.1, G59.2, G59.3
Control Mode (Group 13)	G61, G61.1, G64
Spindle Speed Mode (Group 14)	G96, G97
Lathe Diameter Mode (Group 15)	G7, G8

Table 6.7: M-Code Modal Groups

Modal Group Meaning	Member Words
Stopping (Group 4)	M0, M1, M2, M30, M60
On/Off I/O (Group 5)	FIXME M6 Tn
Toolchange (Group 6)	M6 Tn
Spindle (Group 7)	M3, M4, M5
Coolant (Group 8)	(M7 M8 can both be on), M9
Override Switches (Group 9)	M48, M49
User Defined (Group 10)	M100-M199

For several modal groups, when a machining center is ready to accept commands, one member of the group must be in effect. There are default settings for these modal groups. When the machining center is turned on or otherwise re-initialized, the default values are automatically in effect.

Group 1, the first group on the table, is a group of G-codes for motion. One of these is always in effect. That one is called the current motion mode.

It is an error to put a G-code from group 1 and a G-code from group 0 on the same line if both of them use axis words. If an axis word-using G-code from group 1 is implicitly in effect on a line (by having been activated on an earlier line), and a group 0 G-code that uses axis words appears on the line, the activity of the group 1 G-code is suspended for that line. The axis word-using G-codes from group 0 are G10, G28, G30, G52 and G92.

It is an error to include any unrelated words on a line with *O*- flow control.

6.2.14 Comments

Comments are purely informative and have no influence on machine behaviour.

Comments can be added to lines of G-code to help clear up the intention of the programmer. Comments can be embedded in a line using parentheses () or for the remainder of a line using a semi-colon. The semi-colon is not treated as the start of a comment when enclosed in parentheses.

Comments may appear between words, but not between words and their corresponding parameter. So, *S100(set speed)F200(feed)* is OK while *S(speed)100F(feed)* is not.

Here is an example of a commented program:

```
G0 (Rapid to start) X1 Y1
G0 X1 Y1 (Rapid to start; but don't forget the coolant)
M2 ; End of program.
```

There are several *active* comments which look like comments but cause some action, like *(debug,...)* or *(print,...)*. If there are several comments on a line, only the last comment will be interpreted according to these rules. Hence, a normal comment following an active comment will in effect disable the active comment. For example, *(foo) (debug,#1)* will print the value of parameter *#1*, however *(debug,#1)(foo)* will not.

A comment introduced by a semicolon is by definition the last comment on that line, and will always be interpreted for active comment syntax.

Note

Inline comments on O words should not be used see the O Code [comments](#) section for more information.

6.2.15 Messages

- *(MSG,)* - displays message if *MSG* appears after the left parenthesis and before any other printing characters. Variants of *MSG* which include white space and lower case characters are allowed. The rest of the characters before the right parenthesis are considered to be a message. Messages should be displayed on the message display device of the user interface if provided.

Message Example

```
(MSG, This is a message)
```

6.2.16 Probe Logging

- *(PROBEOPEN filename.txt)* - will open filename.txt and store the 9-number coordinate consisting of XYZABCUVW of each successful straight probe in it.
- *(PROBECLOSE)* - will close the open probelog file.

For more information on probing see the [G38](#) section.

6.2.17 Logging

- *(LOGOPEN,filename.txt)* - opens the named log file. If the file already exists, it is truncated.
- *(LOGAPPEND,filename)* - opens the named log file. If the file already exists, the data is appended.
- *(LOGCLOSE)* - closes an open log file.
- *(LOG,)* - everything past the , is written to the log file if it is open. Supports expansion of parameters as described below.

Examples of logging are in *nc_files/examples/smartprobe.ngc* and in *nc_files/ngcgui_lib/rectangle_probe.ngc* sample G-code files.

6.2.18 Debug Messages

- (*DEBUG*,) - displays a message like (*MSG*,) with the addition of special handling for comment parameters as described below.

6.2.19 Print Messages

- (*PRINT*,) - messages are output to *stderr* with special handling for comment parameters as described below.

6.2.20 Comment Parameters

In the *DEBUG*, *PRINT* and *LOG* comments, the values of parameters in the message are expanded. For example: to print a named global variable to *stderr* (the default console window).

Parameters Example

```
(print,endmill dia = #<_endmill_dia>)  
(print,value of variable 123 is: #123)
```

Inside the above types of comments, sequences like *#123* are replaced by the value of the parameter 123. Sequences like *#<named parameter>* are replaced by the value of the named parameter. Named parameters will have white space removed from them. So, *#<named parameter>* will be converted to *#<namedparameter>*.

Parameter numbers can be formatted, e.g.:

```
(DEBUG, value = %d#<some_value>)
```

will print the value rounded to an integer.

- *%lf* is default if there is no formatting string.
- *%d* = 0 decimals
- *%f* = four decimals
- *%.xf* = x (0-9) number of decimals

The formatting will be performed on all parameters in the same line unless changed, i.e., multiple formatting is allowed in one line.

The formatting string does not need to be right beside the parameter.

If the formatting string is created with the wrong pattern it will be printed as characters.

6.2.21 File Requirements

A G-code file must contain one or more lines of G-code and be terminated with a [Program End](#). Any G-code past the program end is not evaluated.

If a program end code is not used a pair of percent signs *%* with the first percent sign on the first line of the file followed by one or more lines of G-code and a second percent sign. Any code past the second percent sign is not evaluated.

**Warning**

Using % to wrap a G-code file will not do the same thing as using a program end. The machine will be in what ever state the program left it in using %, the spindle and coolant may still be on and things like G90/91 are left as the last program set them. If you don't use a proper preamble the next program could start in a dangerous condition.

Note

The file must be created with a text editor like Gedit and not a word processor like Open Office Word Processor.

6.2.22 File Size

The interpreter and task are carefully written so that the only limit on part program size is disk capacity. The TkLinuxCNC and Axis interface both load the program text to display it to the user, though, so RAM becomes a limiting factor. In Axis, because the preview plot is drawn by default, the redraw time also becomes a practical limit on program size. The preview can be turned off in Axis to speed up loading large part programs. In Axis sections of the preview can be turned off using [preview control](#) comments.

6.2.23 G-Code Order of Execution

The order of execution of items on a line is defined not by the position of each item on the line, but by the following list:

- O-word commands (optionally followed by a comment but no other words allowed on the same line)
 - Comment (including message)
 - Set feed rate mode (G93, G94).
 - Set feed rate (F).
 - Set spindle speed (S).
 - Select tool (T).
 - HAL pin I/O (M62-M68).
 - Change tool (M6) and Set Tool Number (M61).
 - Spindle on or off (M3, M4, M5).
 - Save State (M70, M73), Restore State (M72), Invalidate State (M71).
 - Coolant on or off (M7, M8, M9).
 - Enable or disable overrides (M48, M49,M50,M51,M52,M53).
 - User-defined Commands (M100-M199).
 - Dwell (G4).
 - Set active plane (G17, G18, G19).
 - Set length units (G20, G21).
 - Cutter radius compensation on or off (G40, G41, G42)
-

- Cutter length compensation on or off (G43, G49)
- Coordinate system selection (G54, G55, G56, G57, G58, G59, G59.1, G59.2, G59.3).
- Set path control mode (G61, G61.1, G64)
- Set distance mode (G90, G91).
- Set retract mode (G98, G99).
- Go to reference location (G28, G30) or change coordinate system data (G10) or set axis offsets (G52, G92, G92.1, G92.2, G94).
- Perform motion (G0 to G3, G33, G38.n, G73, G76, G80 to G89), as modified (possibly) by G53.
- Stop (M0, M1, M2, M30, M60).

6.2.24 G-Code Best Practices

Use an appropriate decimal precision Use at least 3 digits after the decimal when milling in millimeters, and at least 4 digits after the decimal when milling in inches.

In particular, tolerance checks of arcs are done for .001 and .0001 according to the active units.

Use consistent white space G-code is most legible when at least one space appears before words. While it is permitted to insert white space in the middle of numbers, there is no reason to do so.

Use Center-format arcs Center-format arcs (which use *I- J- K-* instead of *R-*) behave more consistently than R-format arcs, particularly for included angles near 180 or 360 degrees.

Use a Preamble set modal groups When correct execution of your program depends on modal settings, be sure to set them at the beginning of the part program. Modes can carry over from previous programs and from the MDI commands.

Example Preamble for a Mill

```
G17 G20 G40 G49 G54 G80 G90 G94
```

G17 use XY plane, G20 inch mode, G40 cancel diameter compensation, G49 cancel length offset, G54 use coordinate system 1, G80 cancel canned cycles, G90 absolute distance mode, G94 feed/minute mode.

Perhaps the most critical modal setting is the distance units—If you do not include G20 or G21, then different machines will mill the program at different scales. Other settings, such as the return mode in canned cycles may also be important.

Don't put too many things on one line Ignore everything in section [Order of Execution](#), and instead write no line of code that is the slightest bit ambiguous.

Don't set & use a parameter on the same line Don't use and set a parameter on the same line, even though the semantics are well defined. Updating a variable to a new value, such as `#1=[#1+#2]` is OK.

Don't use line numbers Line numbers offer no benefits. When line numbers are reported in error messages, the numbers refer to the line number in the file, not the N-word value.

When several coordinate systems are moved Consider using the inverse time speed mode.

Because the meaning of an *F* word in meters per minute varies depending on the type of axis to be moved and because the amount of removed material does not depend only on the feed rate, it can be simpler to use G93, inverse speed of time, to achieve the removal of desired material.

6.2.25 Linear and Rotary Axis

Because the meaning of an F-word in feed-per-minute mode varies depending on which axes are commanded to move, and because the amount of material removed does not depend only on the feed rate, it may be easier to use G93 inverse time feed mode to achieve the desired material removal rate.

6.2.26 Common Error Messages

- *G-code out of range* - A G-code greater than G99 was used, the scope of G codes in LinuxCNC is 0 - 99. Not every number between 0 and 99 is a valid G-code.
- *Unknown G-code used* - A G-code was used that is not part of the LinuxCNC G-code language.
- *i,j,k word with no Gx to use it* - i, j and k words must be used on the same line as the G-code.
- *Cannot use axis values without a G-code that uses them* - Axis values can not be used on a line without either a modal G-code in effect or a G-code on the same line.
- *File ended with no percent sign or program end* - Every G-code file must end in a M2 or M30 or be wrapped with the percent sign %.

6.3 G-Codes

6.3.1 Conventions

Conventions used in this section

In the G-code prototypes the hyphen (-) stands for a real value and (<>) denotes an optional item.

If *L-* is written in a prototype the - will often be referred to as the *L number*, and so on for any other letter.

In the G-code prototypes the word *axes* stands for any axis as defined in your configuration.

An optional value will be written like this <*L-*>.

A real value may be:

- An explicit number, *4*
- An expression, *[2+2]*
- A parameter value, *#88*
- A unary function value, *acos[0]*

In most cases, if *axis* words are given (any or all of *X Y Z A B C U V W*), they specify a destination point.

Axis numbers are in the currently active coordinate system, unless explicitly described as being in the absolute coordinate system.

Where axis words are optional, any omitted axes will retain their original value.

Any items in the G-code prototypes not explicitly described as optional are required.

The values following letters are often given as explicit numbers. Unless stated otherwise, the explicit numbers can be real values. For example, *G10 L2* could equally well be written *G[2*5] L[1+1]*. If the value of parameter 100 were 2, *G10 L#100* would also mean the same.

If *L-* is written in a prototype the - will often be referred to as the *L number*, and so on for any other letter.

6.3.2 G-Code Quick Reference Table

Code	Description
G0	Coordinated Motion at Rapid Rate
G1	Coordinated Motion at Feed Rate
G2 G3	Coordinated Helical Motion at Feed Rate
G4	Dwell
G5	Cubic Spline
G5.1	Quadratic B-Spline
G5.2	NURBS, add control point
G7	Diameter Mode (lathe)
G8	Radius Mode (lathe)
G10 L0	Reload Tool Table Data
G10 L1	Set Tool Table Entry
G10 L10	Set Tool Table, Calculated, Workpiece
G10 L11	Set Tool Table, Calculated, Fixture
G10 L2	Coordinate System Origin Setting
G10 L20	Coordinate System Origin Setting Calculated
G17 - G19.1	Plane Select
G20 G21	Set Units of Measure
G28 - G28.1	Go to Predefined Position
G30 - G30.1	Go to Predefined Position
G33	Spindle Synchronized Motion
G33.1	Rigid Tapping
G38.2 - G38.5	Probing
G40	Cancel Cutter Compensation
G41 G42	Cutter Compensation
G41.1 G42.1	Dynamic Cutter Compensation
G43	Use Tool Length Offset from Tool Table
G43.1	Dynamic Tool Length Offset
G43.2	Apply additional Tool Length Offset
G49	Cancel Tool Length Offset
G52	Local Coordinate System Offset
G53	Move in Machine Coordinates
G54-G59.3	Select Coordinate System (1 - 9)
G61	Exact Path Mode
G61.1	Exact Stop Mode
G64	Path Control Mode with Optional Tolerance
G70	Lathe finishing cycle
G71-G72	Lathe roughing cycle
G73	Drilling Cycle with Chip Breaking
G74	Left-hand Tapping Cycle with Dwell
G76	Multi-pass Threading Cycle (Lathe)
G80	Cancel Motion Modes
G81	Drilling Cycle
G82	Drilling Cycle with Dwell
G83	Drilling Cycle with Peck
G84	Right-hand Tapping Cycle with Dwell
G85	Boring Cycle, No Dwell, Feed Out
G86	Boring Cycle, Stop, Rapid Out
G87	Back-boring Cycle' (not yet implemented)'
G88	Boring Cycle, Stop, Manual Out (<i>not yet implemented</i>)
G89	Boring Cycle, Dwell, Feed Out
G90 G91	Distance Mode
G90.1 G91.1	Arc Distance Mode

Code	Description
G92	Coordinate System Offset
G92.1 G92.2	Cancel G92 Offsets
G92.3	Restore G92 Offsets
G93 G94 G95	Feed Modes
G96 G97	Spindle Control Mode, Constant Surface vs Rotation Speed (IPM or m/min vs RPM)
G98 G99	Canned Cycle Z Retract Mode

6.3.3 G0 Rapid Move

G0 <axes>

For rapid motion, program *G0 axes*, where all the axis words are optional. The *G0* is optional if the current motion mode is *G0*. This will produce coordinated motion to the destination point at the maximum rapid rate (or slower). *G0* is typically used as a positioning move.

6.3.3.1 Rapid Velocity Rate

The MAX_VELOCITY setting in the INI file [TRAJ] section defines the maximum rapid traverse rate. The maximum rapid traverse rate can be higher than the individual axes MAX_VELOCITY setting during a coordinated move. The maximum rapid traverse rate can be slower than the MAX_VELOCITY setting in the [TRAJ] section if an axis MAX_VELOCITY or trajectory constraints limit it.

G0 Example

```
G90 (set absolute distance mode)
G0 X1 Y-2.3 (Rapid linear move from current location to X1 Y-2.3)
M2 (end program)
```

- See [G90](#) & [M2](#) sections for more information.

If cutter compensation is active, the motion will differ from the above; see the [Cutter Compensation](#) section.

If *G53* is programmed on the same line, the motion will also differ; see the [G53](#) section for more information.

The path of a *G0* rapid motion can be rounded at direction changes and depends on the [trajectory control](#) settings and maximum acceleration of the axes.

It is an error if:

- An axis letter is without a real value.
- An axis letter is used that is not configured.

6.3.4 G1 Linear Move

G1 axes

For linear (straight line) motion at programmed [feed rate](#) (for cutting or not), program *G1 'axes'*, where all the axis words are optional. The *G1* is optional if the current motion mode is *G1*. This will produce coordinated motion to the destination point at the current feed rate (or slower).

G1 Example

```
G90 (set absolute distance mode)
G1 X1.2 Y-3 F10 (linear move at a feed rate of 10 from current position to X1.2 Y-3)
Z-2.3 (linear move at same feed rate from current position to Z-2.3)
Z1 F25 (linear move at a feed rate of 25 from current position to Z1)
M2 (end program)
```

- See [G90](#) & [F](#) & [M2](#) sections for more information.

If cutter compensation is active, the motion will differ from the above; see the [Cutter Compensation](#) section.

If [G53](#) is programmed on the same line, the motion will also differ; see the [G53](#) section for more information.

It is an error if:

- No feed rate has been set.
- An axis letter is without a real value.
- An axis letter is used that is not configured

6.3.5 G2, G3 Arc Move

```
G2 or G3 axes offsets (center format)
G2 or G3 axes R- (radius format)
G2 or G3 offsets|R- <P-> (full circles)
```

A circular or helical arc is specified using either [G2](#) (clockwise arc) or [G3](#) (counterclockwise arc) at the current [feed rate](#). The direction (CW, CCW) is as viewed from the positive end of the axis about which the circular motion occurs.

The axis of the circle or helix must be parallel to the X, Y, or Z axis of the machine coordinate system. The axis (or, equivalently, the plane perpendicular to the axis) is selected with [G17](#) (Z-axis, XY-plane), [G18](#) (Y-axis, XZ-plane), or [G19](#) (X-axis, YZ-plane). Planes [17.1](#), [18.1](#), and [19.1](#) are not currently supported. If the arc is circular, it lies in a plane parallel to the selected plane.

To program a helix, include the axis word perpendicular to the arc plane, for example, if in the [G17](#) plane, include a [Z](#) word. This will cause the [Z](#) axis to move to the programmed value during the circular [XY](#) motion.

To program an arc that gives more than one full turn, use the [P](#) word specifying the number of full turns plus the programmed arc. The [P](#) word must be an integer. If [P](#) is unspecified, the behavior is as if [P1](#) was given that is, only one full or partial turn will result. For example, if a 180 degree arc is programmed with a [P2](#), the resulting motion will be 1 1/2 rotations. For each [P](#) increment above 1 an extra full circle is added to the programmed arc. Multi turn helical arcs are supported and give motion useful for milling holes or threads.



Warning

If the pitch of the helix is very small (less than the [naive CAM tolerance](#)) then the helix might be converted into a straight line. [Bug #222](#)

If a line of code makes an arc and includes rotary axis motion, the rotary axes turn at a constant rate so that the rotary motion starts and finishes when the [XYZ](#) motion starts and finishes. Lines of this sort are hardly ever programmed.

If cutter compensation is active, the motion will differ from the above; see the [Cutter Compensation](#) section.

The arc center is absolute or relative as set by [G90.1](#) or [G91.1](#) respectively.

Two formats are allowed for specifying an arc: Center Format and Radius Format.

It is an error if:

- No feed rate has been set.
- The P word is not an integer.

6.3.5.1 Center Format Arcs

Center format arcs are more accurate than radius format arcs and are the preferred format to use.

The end point of the arc along with the offset to the center of the arc from the current location are used to program arcs that are less than a full circle. It is OK if the end point of the arc is the same as the current location.

The offset to the center of the arc from the current location and optionally the number of turns are used to program full circles.

When programming arcs an error due to rounding can result from using a precision of less than 4 decimal places (0.0000) for inch and less than 3 decimal places (0.000) for millimeters.

Incremental Arc Distance Mode Arc center offsets are a relative distance from the start location of the arc. Incremental Arc Distance Mode is default.

One or more axis words and one or more offsets must be programmed for an arc that is less than 360 degrees.

No axis words and one or more offsets must be programmed for full circles. The *P* word defaults to 1 and is optional.

For more information on 'Incremental Arc Distance Mode see the [G91.1](#) section.

Absolute Arc Distance Mode Arc center offsets are the absolute distance from the current 0 position of the axis.

One or more axis words and *both* offsets must be programmed for arcs less than 360 degrees.

No axis words and both offsets must be programmed for full circles. The *P* word defaults to 1 and is optional.

For more information on 'Absolute Arc Distance Mode see the [G90.1](#) section.

XY-plane (G17)

```
G2 or G3 <X- Y- Z- I- J- P->
```

- Z - helix
- I - X offset
- J - Y offset
- P - number of turns

XZ-plane (G18)

```
G2 or G3 <X- Z- Y- I- K- P->
```

- *Y* - helix
- *I* - X offset
- *K* - Z offset
- *P* - number of turns

YZ-plane (G19)

G2 or G3 <Y- Z- X- J- K- P->

- *X* - helix
- *J* - Y offset
- *K* - Z offset
- *P* - number of turns

It is an error if:

- No feed rate is set with the *F* word.
- No offsets are programmed.
- When the arc is projected on the selected plane, the distance from the current point to the center differs from the distance from the end point to the center by more than (.05 inch/.5 mm) OR ((.0005 inch/.005mm) AND .1% of radius).

Deciphering the Error message *Radius to end of arc differs from radius to start*:

- *start* - the current position
- *center* - the center position as calculated using the *i*, *j*, or *k* words
- *end* - the programmed end point
- *r1* - radius from the start position to the center
- *r2* - radius from the end position to the center

6.3.5.2 Center Format Examples

Calculating arcs by hand can be difficult at times. One option is to draw the arc with a CAD program to get the coordinates and offsets. Keep in mind the tolerance mentioned above, you may have to change the precision of your CAD program to get the desired results. Another option is to calculate the coordinates and offset using formulas. As you can see in the following figures a triangle can be formed from the current position the end position and the arc center.

In the following figure you can see the start position is X0 Y0, the end position is X1 Y1. The arc center position is at X1 Y0. This gives us an offset from the start position of 1 in the X axis and 0 in the Y axis. In this case only an *I* offset is needed.

G2 Example Line

```
G0 X0 Y0
G2 X1 Y1 I1 F10 (clockwise arc in the XY plane)
```

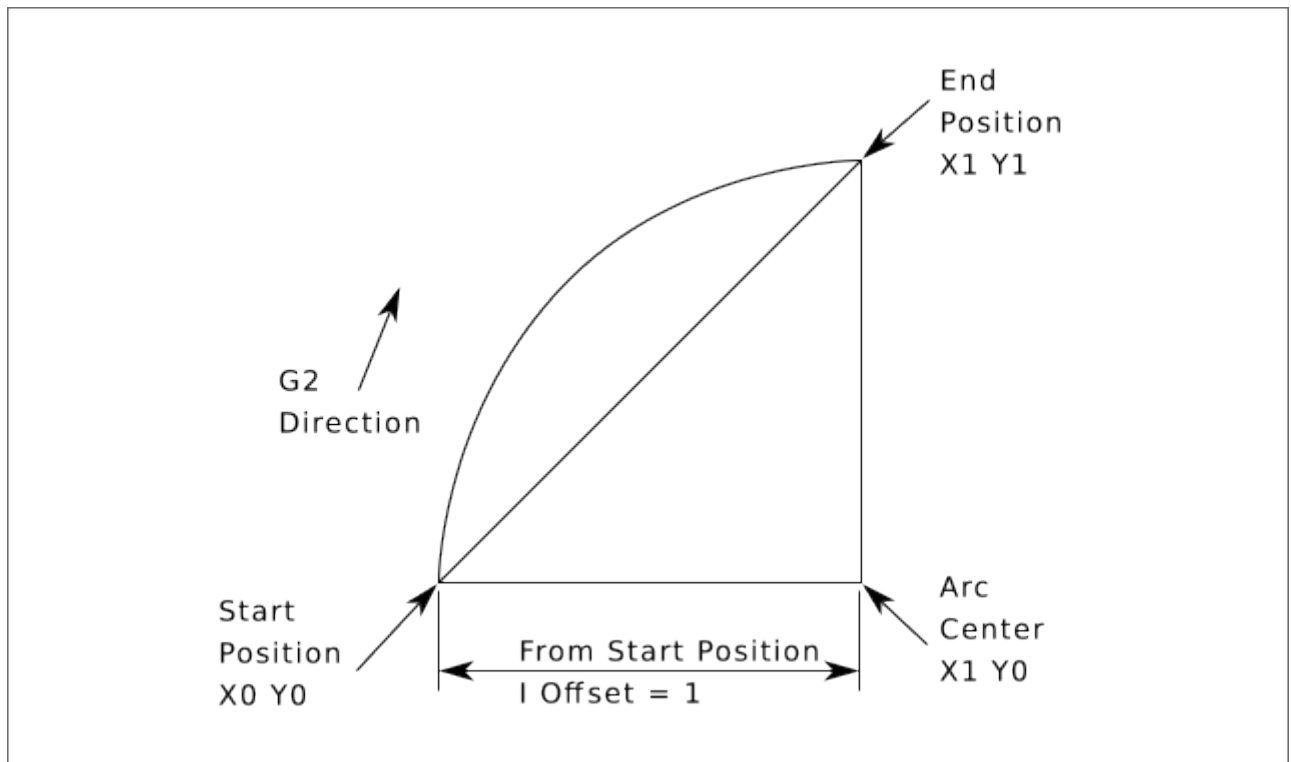


Figure 6.4: G2 Example

In the next example we see the difference between the offsets for Y if we are doing a G2 or a G3 move. For the G2 move the start position is $X0\ Y0$, for the G3 move it is $X0\ Y1$. The arc center is at $X1\ Y0.5$ for both moves. The G2 move the J offset is 0.5 and the G3 move the J offset is -0.5.

G2-G3 Example Line

```
G0 X0 Y0
G2 X0 Y1 I1 J0.5 F25 (clockwise arc in the XY plane)
G3 X0 Y0 I1 J-0.5 F25 (counterclockwise arc in the XY plane)
```

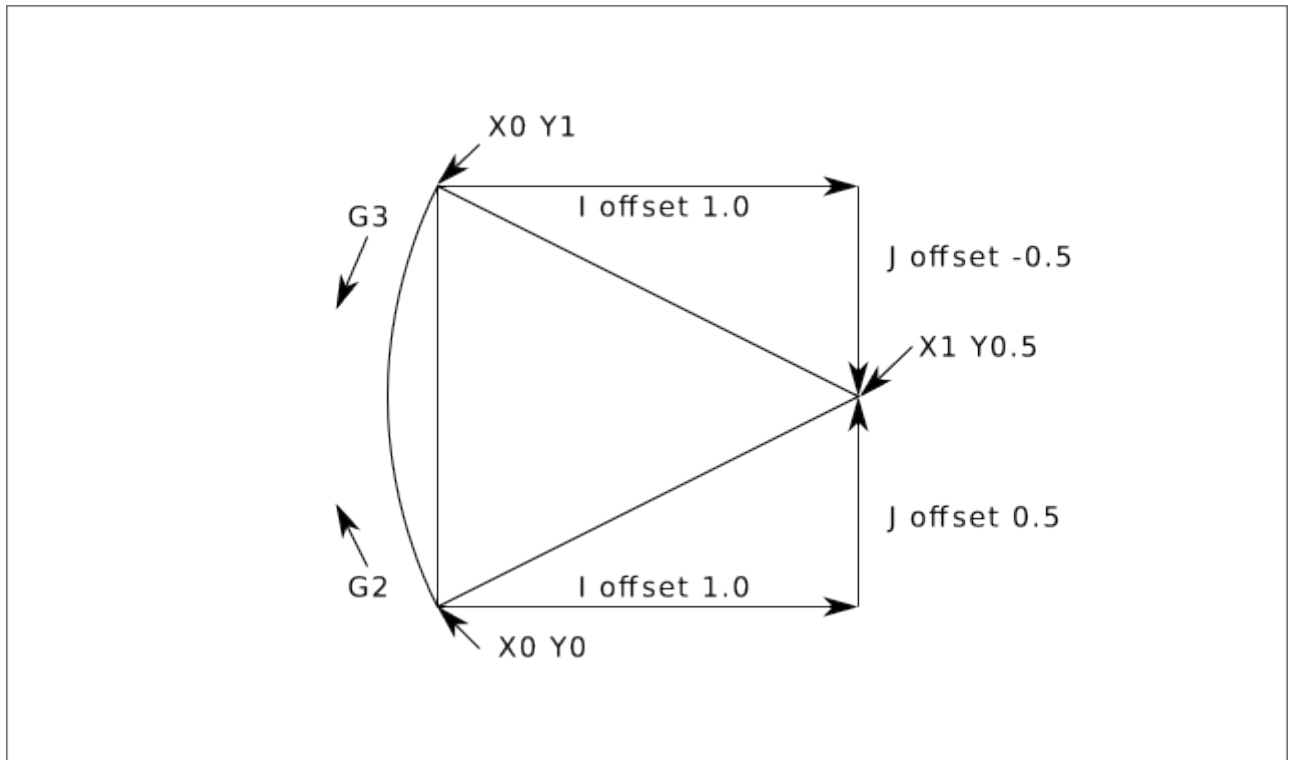


Figure 6.5: G2-G3 Example

In the next example we show how the arc can make a helix in the Z axis by adding the Z word.

G2 Example Helix

```
G0 X0 Y0 Z0
G17 G2 X10 Y16 I3 J4 Z-1 (helix arc with Z added)
```

In the next example we show how to make more than one turn using the P word.

P word Example

```
G0 X0 Y0 Z0
G2 X0 Y1 Z-1 I1 J0.5 P2 F25
```

In the center format, the radius of the arc is not specified, but it may be found easily as the distance from the center of the circle to either the current point or the end point of the arc.

6.3.5.3 Radius Format Arcs

```
G2 or G3 axes R- <P->
```

- R - radius from current position

It is not good practice to program radius format arcs that are nearly full circles or nearly semicircles because a small change in the location of the end point will produce a much larger change in the location of the center of the circle (and, hence, the middle of the arc). The magnification effect is large enough that rounding error in a number can produce out-of-tolerance cuts. For instance, a 1% displacement of the endpoint of a 180 degree arc produced a 7% displacement of the point 90 degrees

along the arc. Nearly full circles are even worse. Other size arcs (in the range tiny to 165 degrees or 195 to 345 degrees) are OK.

In the radius format, the coordinates of the end point of the arc in the selected plane are specified along with the radius of the arc. Program *G2 axes R-* (or use *G3* instead of *G2*). *R* is the radius. The axis words are all optional except that at least one of the two words for the axes in the selected plane must be used. The *R* number is the radius. A positive radius indicates that the arc turns through less than 180 degrees, while a negative radius indicates a turn of more than 180 degrees. If the arc is helical, the value of the end point of the arc on the coordinate axis parallel to the axis of the helix is also specified.

It is an error if:

- both of the axis words for the axes of the selected plane are omitted
- the end point of the arc is the same as the current point.

G2 Example Line

```
G17 G2 X10 Y15 R20 Z5 (radius format with arc)
```

The above example makes a clockwise (as viewed from the positive Z-axis) circular or helical arc whose axis is parallel to the Z-axis, ending where X=10, Y=15, and Z=5, with a radius of 20. If the starting value of Z is 5, this is an arc of a circle parallel to the XY-plane; otherwise it is a helical arc.

6.3.6 G4 Dwell

```
G4 P-
```

- *P* - seconds to dwell (floating point)

The *P* number is the time in seconds that all axes will remain unmoving. The *P* number is a floating point number so fractions of a second may be used. *G4* does not affect spindle, coolant and any I/O.

G4 Example Line

```
G4 P0.5 (wait for 0.5 seconds before proceeding)
```

It is an error if:

- the *P* number is negative or not specified.

6.3.7 G5 Cubic Spline

```
G5 X- Y- <I- J-> P- Q-
```

- *I* - X incremental offset from start point to first control point
 - *J* - Y incremental offset from start point to first control point
 - *P* - X incremental offset from end point to second control point
 - *Q* - Y incremental offset from end point to second control point
-

G5 creates a cubic B-spline in the XY plane with the X and Y axes only. P and Q must both be specified for every G5 command.

For the first G5 command in a series of G5 commands, I and J must both be specified. For subsequent G5 commands, either both I and J must be specified, or neither. If I and J are unspecified, the starting direction of this cubic will automatically match the ending direction of the previous cubic (as if I and J are the negation of the previous P and Q).

For example, to program a curvy N shape:

G5 Sample initial cubic spline

```
G90 G17
G0 X0 Y0
G5 I0 J3 P0 Q-3 X1 Y1
```

A second curvy N that attaches smoothly to this one can now be made without specifying I and J:

G5 Sample subsequent cubic spline

```
G5 P0 Q-3 X2 Y2
```

It is an error if:

- P and Q are not both specified.
- Just one of I or J are specified.
- I or J are unspecified in the first of a series of G5 commands.
- An axis other than X or Y is specified.
- The active plane is not G17.

6.3.8 G5.1 Quadratic Spline

```
G5.1 X- Y- I- J-
```

- *I* - X incremental offset from start point to control point
- *J* - Y incremental offset from start point to control point

G5.1 creates a quadratic B-spline in the XY plane with the X and Y axis only. Not specifying I or J gives zero offset for the unspecified axis, so one or both must be given.

For example, to program a parabola, through the origin, from X-2 Y4 to X2 Y4:

G5.1 Sample quadratic spline

```
G90 G17
G0 X-2 Y4
G5.1 X2 I2 J-8
```

It is an error if:

- both I and J offset are unspecified or zero
 - An axis other than X or Y is specified
 - The active plane is not G17
-

6.3.9 G5.2 G5.3 NURBS Block

```
G5.2 <P-> <X- Y-> <L->
X- Y- <P->
...
G5.3
```



Warning

G5.2, G5.3 is experimental and not fully tested.

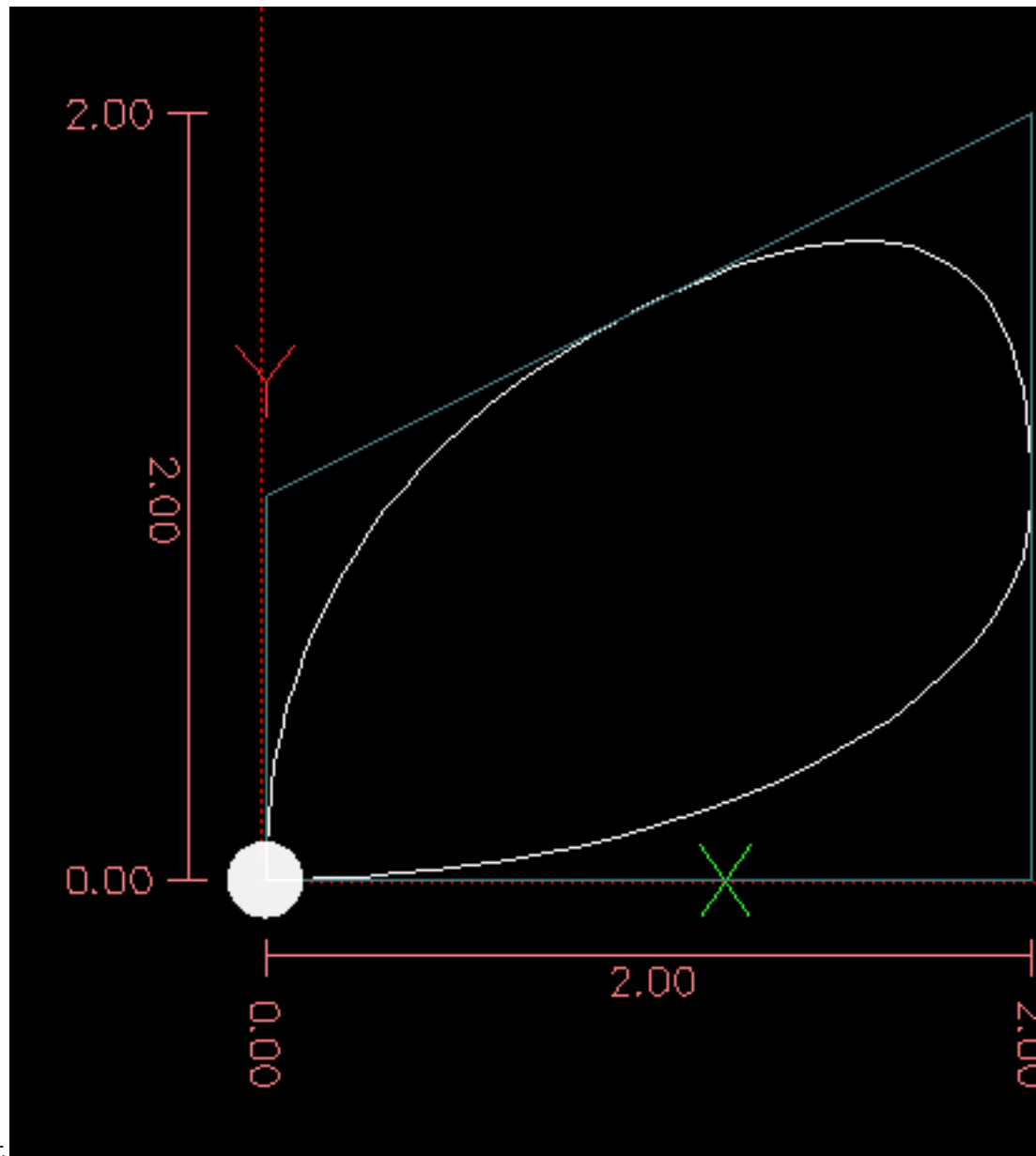
G5.2 is for opening the data block defining a NURBS and G5.3 for closing the data block. In the lines between these two codes the curve control points are defined with both their related *weights* (P) and the parameter (L) which determines the order of the curve.

The current coordinate, before the first G5.2 command, is always taken as the first NURBS control point. To set the weight for this first control point, first program G5.2 P- without giving any X Y.

The default weight if P is unspecified is 1. The default order if L is unspecified is 3.

G5.2 Example

```
G0 X0 Y0 (rapid move)
F10 (set feed rate)
G5.2 P1 L3
    X0 Y1 P1
    X2 Y2 P1
    X2 Y0 P1
    X0 Y0 P2
G5.3
; The rapid moves show the same path without the NURBS Block
G0 X0 Y1
    X2 Y2
    X2 Y0
    X0 Y0
M2
```



Sample NURBS Output

More information on NURBS can be found here:

<http://wiki.linuxcnc.org/cgi-bin/wiki.pl?NURBS>

6.3.10 G7 Lathe Diameter Mode

G7

Program G7 to enter the diameter mode for axis X on a lathe. When in the diameter mode the X axis moves on a lathe will be 1/2 the distance to the center of the lathe. For example X1 would move the cutter to 0.500" from the center of the lathe thus giving a 1" diameter part.

6.3.11 G8 Lathe Radius Mode

G8

Program G8 to enter the radius mode for axis X on a lathe. When in Radius mode the X axis moves on a lathe will be the distance from the center. Thus a cut at X1 would result in a part that is 2" in diameter. G8 is default at power up.

6.3.12 G10 L0 Reload Tool Table Data

G10 L0

G10 L0 reload all tool table data. Requires that there is no current tool loaded in spindle.

Note

When using G10 L0, tool parameters (#5401-#5413) will be updated immediately and any altered tool diameters will be used for subsequent G41,42 cutter radius compensation commands. Existing G43 tool length compensation values will remain in effect until updated by new G43 commands.

6.3.13 G10 L1 Set Tool Table

G10 L1 P- axes <R- I- J- Q->

- *P* - tool number
- *R* - radius of tool
- *I* - front angle (lathe)
- *J* - back angle (lathe)
- *Q* - orientation (lathe)

G10 L1 sets the tool table for the *P* tool number to the values of the words.

A valid G10 L1 rewrites and reloads the tool table for the specified tool.

G10 L1 Example Line

```
G10 L1 P1 Z1.5 (set tool 1 Z offset from the machine origin to 1.5)
G10 L1 P2 R0.015 Q3 (lathe example setting tool 2 radius to 0.015 and orientation to 3)
```

It is an error if:

- Cutter Compensation is on
- The *P* number is unspecified
- The *P* number is not a valid tool number from the tool table
- The *P* number is 0

For more information on cutter orientation used by the *Q* word, see the [Lathe Tool Orientation](#) diagram.

6.3.14 G10 L2 Set Coordinate System

G10 L2 P- <axes R->

- *P* - coordinate system (0-9)
- *R* - rotation about the Z axis

G10 L2 offsets the origin of the axes in the coordinate system specified to the value of the axis word. The offset is from the machine origin established during homing. The offset value will replace any current offsets in effect for the coordinate system specified. Axis words not used will not be changed.

Program P0 to P9 to specify which coordinate system to change.

Table 6.8: Coordinate System

P Value	Coordinate System	G-code
0	Active	n/a
1	1	G54
2	2	G55
3	3	G56
4	4	G57
5	5	G58
6	6	G59
7	7	G59.1
8	8	G59.2
9	9	G59.3

Optionally program R to indicate the rotation of the XY axis around the Z axis. The direction of rotation is CCW as viewed from the positive end of the Z axis.

All axis words are optional.

Being in incremental distance mode ([G91](#)) has no effect on *G10 L2*.

Important Concepts:

- G10 L2 Pn does not change from the current coordinate system to the one specified by P, you have to use G54-59.3 to select a coordinate system.
- When a rotation is in effect jogging an axis will only move that axis in a positive or negative direction and not along the rotated axis.
- If a G52 local offset or G92 origin offset was in effect before *G10 L2*, it will continue to be in effect afterwards.
- When programming a coordinate system with R, any G52 or G92 will be applied **after** the rotation.
- The coordinate system whose origin is set by a *G10* command may be active or inactive at the time the *G10* is executed. If it is currently active, the new coordinates take effect immediately.

It is an error if:

- The P number does not evaluate to an integer in the range 0 to 9.

- An axis is programmed that is not defined in the configuration.

G10 L2 Example Line

```
G10 L2 P1 X3.5 Y17.2
```

In the above example the origin of the first coordinate system (the one selected by *G54*) is set to be *X*=3.5 and *Y*=17.2. Because only *X* and *Y* are specified, the origin point is only moved in *X* and *Y*; the other coordinates are not changed.

G10 L2 Example Line

```
G10 L2 P1 X0 Y0 Z0 (clear offsets for X,Y & Z axes in coordinate system 1)
```

The above example sets the XYZ coordinates of the coordinate system 1 to the machine origin.

The coordinate system is described in the [Coordinate System](#) section.

6.3.15 G10 L10 Set Tool Table

```
G10 L10 P- axes <R- I- J- Q->
```

- *P* - tool number
- *R* - radius of tool
- *I* - front angle (lathe)
- *J* - back angle (lathe)
- *Q* - orientation (lathe)

G10 L10 changes the tool table entry for tool *P* so that if the tool offset is reloaded, with the machine in its current position and with the current *G5x* and *G52/G92* offsets active, the current coordinates for the given axes will become the given values. The axes that are not specified in the G10 L10 command will not be changed. This could be useful with a probe move as described in the [G38](#) section.

G10 L10 Example

```
T1 M6 G43 (load tool 1 and tool length offsets)
G10 L10 P1 Z1.5 (set the current position for Z to be 1.5)
G43 (reload the tool length offsets from the changed tool table)
M2 (end program)
```

- See [T](#) & [M6](#), and [G43/G43.1](#) sections for more information.

It is an error if:

- Cutter Compensation is on
 - The *P* number is unspecified
 - The *P* number is not a valid tool number from the tool table
 - The *P* number is 0
-

6.3.16 G10 L11 Set Tool Table

G10 L11 P- axes <R- I- J- Q->

- *P* - tool number
- *R* - radius of tool
- *I* - front angle (lathe)
- *J* - back angle (lathe)
- *Q* - orientation (lathe)

G10 L11 is just like G10 L10 except that instead of setting the entry according to the current offsets, it is set so that the current coordinates would become the given value if the new tool offset is reloaded and the machine is placed in the G59.3 coordinate system without any G52/G92 offset active.

This allows the user to set the G59.3 coordinate system according to a fixed point on the machine, and then use that fixture to measure tools without regard to other currently-active offsets. It is an error if:

- Cutter Compensation is on
- The P number is unspecified
- The P number is not a valid tool number from the tool table
- The P number is 0

6.3.17 G10 L20 Set Coordinate System

G10 L20 P- axes

- *P* - coordinate system (0-9)

G10 L20 is similar to G10 L2 except that instead of setting the offset/entry to the given value, it is set to a calculated value that makes the current coordinates become the given value.

G10 L20 Example Line

G10 L20 P1 X1.5 (set the X axis current location in coordinate system 1 to 1.5)

It is an error if:

- The P number does not evaluate to an integer in the range 0 to 9.
- An axis is programmed that is not defined in the configuration.

6.3.18 G17 - G19.1 Plane Select

These codes set the current plane as follows:

- *G17* - XY (default)
- *G18* - ZX
- *G19* - YZ
- *G17.1* - UV
- *G18.1* - WU
- *G19.1* - VW

The UV, WU and VW planes do not support arcs.

It is a good idea to include a plane selection in the preamble of each G-code file.

The effects of having a plane selected are discussed in section [G2 G3 Arcs](#) and section [G81 G89](#).

6.3.19 G20, G21 Units

- *G20* - to use inches for length units.
- *G21* - to use millimeters for length units.

It is a good idea to include units in the preamble of each G-code file.

6.3.20 G28, G28.1 Go/Set Predefined Position



Warning

Only use G28 when your machine is homed to a repeatable position and the desired G28 position has been stored with G28.1.

G28 uses the values stored in [parameters](#) 5161-5169 as the X Y Z A B C U V W final point to move to. The parameter values are *absolute* machine coordinates in the native machine *units* as specified in the INI file. All axes defined in the INI file will be moved when a G28 is issued. If no positions are stored with G28.1 then all axes will go to the [machine origin](#).

- *G28* - makes a [rapid move](#) from the current position to the *absolute* position of the values in parameters 5161-5166.
- *G28 axes* - makes a rapid move to the position specified by *axes* including any offsets, then will make a rapid move to the *absolute* position of the values in parameters 5161-5166 for all *axes* specified. Any *axis* not specified will not move.
- *G28.1* - stores the current *absolute* position into parameters 5161-5166.

G28 Example Line

```
G28 Z2.5 (rapid to Z2.5 then to Z location specified in #5163)
```

It is an error if :

- Cutter Compensation is turned on

6.3.21 G30, G30.1 Go/Set Predefined Position



Warning

Only use G30 when your machine is homed to a repeatable position and the desired G30 position has been stored with G30.1.

G30 functions the same as G28 but uses the values stored in [parameters](#) 5181-5189 as the X Y Z A B C U V W final point to move to. The parameter values are *absolute* machine coordinates in the native machine *units* as specified in the INI file. All axes defined in the INI file will be moved when a G30 is issued. If no positions are stored with G30.1 then all axes will go to the [machine origin](#).

Note

G30 parameters will be used to move the tool when a M6 is programmed if TOOL_CHANGE_AT_G30=1 is in the [EMCIO] section of the INI file.

- *G30* - makes a [rapid move](#) from the current position to the *absolute* position of the values in parameters 5181-5189.
- *G30 axes* - makes a rapid move to the position specified by *axes* including any offsets, then will make a rapid move to the *absolute* position of the values in parameters 5181-5189 for all *axes* specified. Any *axis* not specified will not move.
- *G30.1* - stores the current absolute position into parameters 5181-5186.

G30 Example Line

```
G30 Z2.5 (rapid to Z2.5 then to the Z location specified in #5183)
```

It is an error if :

- Cutter Compensation is turned on

6.3.22 G33 Spindle Synchronized Motion

```
G33 X- Y- Z- K- $-
```

- *K* - distance per revolution

For spindle-synchronized motion in one direction, code *G33 X- Y- Z- K-* where *K* gives the distance moved in XYZ for each revolution of the spindle. For instance, if starting at *Z=0*, *G33 Z-1 K.0625* produces a 1 inch motion in Z over 16 revolutions of the spindle. This command might be part of a program to produce a 16TPI thread. Another example in metric, *G33 Z-15 K1.5* produces a movement of 15mm while the spindle rotates 10 times for a thread of 1.5mm.

The (optional) *\$* argument sets which spindle the motion is synchronised to (default is zero). For example *G33 Z10 K1 \$1* will move the spindle in synchrony with the spindle.N.revs HAL pin value.

Spindle-synchronized motion waits for the spindle index and spindle at speed pins, so multiple passes line up. *G33* moves end at the programmed endpoint. *G33* could be used to cut tapered threads or a fusee.

All the axis words are optional, except that at least one must be used.

Note

K follows the drive line described by X- Y- Z-. K is not parallel to the Z axis if X or Y endpoints are used for example when cutting tapered threads.

Technical Info

At the beginning of each G33 pass, LinuxCNC uses the spindle speed and the machine acceleration limits to calculate how long it will take Z to accelerate after the index pulse, and determines how many degrees the spindle will rotate during that time. It then adds that angle to the index position and computes the Z position using the corrected spindle angle. That means that Z will reach the correct position just as it finishes accelerating to the proper speed, and can immediately begin cutting a good thread.

HAL Connections The pin *spindle.N.at-speed* must be set or driven true for the motion to start. Additionally spindle.N.revs must increase by 1 for each revolution of the spindle and the spindle.N.index-enable pin must be connected to an encoder (or resolver) counter which resets index-enable once per rev.

See the Integrators Manual for more information on spindle synchronized motion.

G33 Example

```
G90 (absolute distance mode)
G0 X1 Z0.1 (rapid to position)
S100 M3 (start spindle turning)
G33 Z-2 K0.125 (move Z axis to -2 at a rate to equal 0.125 per revolution)
G0 X1.25 (rapid move tool away from work)
Z0.1 (rapid move to starting Z position)
M2 (end program)
```

- See [G90](#) & [G0](#) & [M2](#) sections for more information.

It is an error if:

- All axis words are omitted.
- The spindle is not turning when this command is executed.
- The requested linear motion exceeds machine velocity limits due to the spindle speed.

6.3.23 G33.1 Rigid Tapping

G33.1 X- Y- Z- K- I- \$-

- K - distance per revolution
- I - optional spindle speed multiplier for faster return move
- \$ - optional spindle selector

**Warning**

For Z only tapping preposition the XY location prior to calling G33.1 and only use a Z word in the G33.1. If the coordinates specified are not the current coordinates when calling G33.1 for tapping the move will not be along the Z axis but will be a coordinated, spindle-synchronized move from the current location to the location specified and back.

For rigid tapping (spindle synchronized motion with return), code *G33.1 X- Y- Z- K-* where *K-* gives the distance moved for each revolution of the spindle.

A rigid tapping move consists of the following sequence:

- A move from the current coordinate to the specified coordinate, synchronized with the selected spindle at the given ratio and starting from the current coordinate with a spindle index pulse.
- When reaching the endpoint, a command to reverse the spindle, and speed up by a factor set by the multiplier (e.g., from clockwise to counterclockwise).
- Continued synchronized motion beyond the specified end coordinate until the spindle actually stops and reverses.
- Continued synchronized motion back to the original coordinate.
- When reaching the original coordinate, a command to reverse the spindle a second time (e.g., from counterclockwise to clockwise).
- Continued synchronized motion beyond the original coordinate until the spindle actually stops and reverses.
- An **unsynchronized** move back to the original coordinate.

Spindle-synchronized motions wait for spindle index, so multiple passes line up. *G33.1* moves end at the original coordinate.

All the axis words are optional, except that at least one must be used.

G33.1 Example

```
G90 (set absolute mode)
G0 X1.000 Y1.000 Z0.100 (rapid move to starting position)
S100 M3 (turn on the spindle, 100 RPM)
G33.1 Z-0.750 K0.05 (rigid tap a 20 TPI thread 0.750 deep)
M2 (end program)
```

- See [G90](#) & [G0](#) & [M2](#) sections for more information.

It is an error if:

- All axis words are omitted.
- The spindle is not turning when this command is executed
- The requested linear motion exceeds machine velocity limits due to the spindle speed

6.3.24 G38.n Straight Probe

G38.n axes

- *G38.2* - probe toward workpiece, stop on contact, signal error if failure
 - *G38.3* - probe toward workpiece, stop on contact
 - *G38.4* - probe away from workpiece, stop on loss of contact, signal error if failure
 - *G38.5* - probe away from workpiece, stop on loss of contact
-

**Important**

You will not be able to use a probe move until your machine has been set up to provide a probe input signal. The probe input signal must be connected to *motion.probe-input* in a .hal file. G38.n uses *motion.probe-input* to determine when the probe has made (or lost) contact. TRUE for probe contact closed (touching), FALSE for probe contact open.

Program *G38.n axes* to perform a straight probe operation. The axis words are optional, except that at least one of them must be used. The axis words together define the destination point that the probe will move towards, starting from the current location. If the probe is not tripped before the destination is reached G38.2 and G38.4 will signal an error.

The tool in the spindle must be a probe or contact a probe switch.

In response to this command, the machine moves the controlled point (which should be at the center of the probe ball) in a straight line at the current [feed rate](#) toward the programmed point. In inverse time feed mode, the feed rate is such that the whole motion from the current point to the programmed point would take the specified time. The move stops (within machine acceleration limits) when the programmed point is reached, or when the requested change in the probe input takes place, whichever occurs first.

Table 6.9: Probing G-Codes

Code	Target State	Move orientation	Error Signal
G38.2	Touched	Toward piece	Yes
G38.3	Touched	Toward piece	No
G38.4	Untouched	From piece	Yes
G38.5	Untouched	From piece	No

After successful probing, parameters #5061 to #5069 will be set to the X, Y, Z, A, B, C, U, V, W coordinates of the location of the controlled point at the time the probe changed state (in the current work coordinate system). After unsuccessful probing, they are set to the coordinates of the programmed point. Parameter 5070 is set to 1 if the probe succeeded and 0 if the probe failed. If the probing operation failed, G38.2 and G38.4 will signal an error by posting a message on screen if the selected GUI supports that. And by halting program execution.

A comment of the form (*PROBEOPEN filename.txt*) will open *filename.txt* and store the 9-number coordinate consisting of XYZABCUVW of each successful straight probe in it. The file must be closed with (*PROBECLOSE*). For more information see the [Comments](#) section.

An example file *smartprobe.ngc* is included (in the examples directory) to demonstrate using probe moves to log to a file the coordinates of a part. The program *smartprobe.ngc* could be used with *ngcgui* with minimal changes.

It is an error if:

- the current point is the same as the programmed point.
- no axis word is used
- cutter compensation is enabled
- the feed rate is zero
- the probe is already in the target state

6.3.25 G40 Compensation Off

- *G40* - turn cutter compensation off. If tool compensation was on the next move must be a linear move and longer than the tool diameter. It is OK to turn compensation off when it is already off.

G40 Example

```
; current location is X1 after finishing cutter compensated move
G40 (turn compensation off)
G0 X1.6 (linear move longer than current cutter diameter)
M2 (end program)
```

See [G0](#) & [M2](#) sections for more information.

It is an error if:

- A G2/G3 arc move is programmed next after a G40.
- The linear move after turning compensation off is less than the tool diameter.

6.3.26 G41, G42 Cutter Compensation

```
G41 <D> (left of programmed path)
G42 <D> (right of programmed path)
```

- *D* - tool number

The *D* word is optional; if there is no *D* word the radius of the currently loaded tool will be used (if no tool is loaded and no *D* word is given, a radius of 0 will be used).

If supplied, the *D* word is the tool number to use. This would normally be the number of the tool in the spindle (in which case the *D* word is redundant and need not be supplied), but it may be any valid tool number.

Note

G41/G42 D0 is a little special. Its behavior is different on random tool changer machines and nonrandom tool changer machines (see the [Tool Change](#) section). On nonrandom tool changer machines, *G41/G42 D0* applies the Tool Length Offset of the tool currently in the spindle, or a TLO of 0 if no tool is in the spindle. On random tool changer machines, *G41/G42 D0* applies the TLO of the tool *T0* defined in the tool table file (or causes an error if *T0* is not defined in the tool table).

To start cutter compensation to the left of the part profile, use G41. G41 starts cutter compensation to the left of the programmed line as viewed from the positive end of the axis perpendicular to the plane.

To start cutter compensation to the right of the part profile, use G42. G42 starts cutter compensation to the right of the programmed line as viewed from the positive end of the axis perpendicular to the plane.

The lead in move must be at least as long as the tool radius. The lead in move can be a rapid move.

Cutter compensation may be performed if the XY-plane or XZ-plane is active.

User M100-M199 commands are allowed when Cutter Compensation is on.

The behavior of the machining center when cutter compensation is on is described in the [Cutter Compensation](#) section along with code examples.

It is an error if:

- The D number is not a valid tool number or 0.
- The YZ plane is active.
- Cutter compensation is commanded to turn on when it is already on.

6.3.27 G41.1, G42.1 Dynamic Cutter Compensation

G41.1 D- <L-> (left of programmed path)
 G42.1 D- <L-> (right of programmed path)

- *D* - cutter diameter
- *L* - tool orientation (see [lathe tool orientation](#))

G41.1 & G42.1 function the same as G41 & G42 with the added scope of being able to program the tool diameter. The L word defaults to 0 if unspecified.

It is an error if:

- The YZ plane is active.
- The L number is not in the range from 0 to 9 inclusive.
- The L number is used when the XZ plane is not active.
- Cutter compensation is commanded to turn on when it is already on.

6.3.28 G43 Tool Length Offset

G43 <H->

- *H* - tool number (optional)
- *G43* - enables tool length compensation. *G43* changes subsequent motions by offsetting the axis coordinates by the length of the offset. *G43* does not cause any motion. The next time a compensated axis is moved, that axis's endpoint is the compensated location.

G43 without an H word uses the currently loaded tool from the last *Tn M6*.

G43 Hn uses the offset for tool n.

Note

G43 H0 is a little special. Its behavior is different on random tool changer machines and nonrandom tool changer machines (see the [Tool Changers](#) section). On nonrandom tool changer machines, *G43 H0* applies the Tool Length Offset of the tool currently in the spindle, or a TLO of 0 if no tool is in the spindle. On random tool changer machines, *G43 H0* applies the TLO of the tool T0 defined in the tool table file (or causes an error if T0 is not defined in the tool table).

G43 H- Example Line

G43 H1 (set tool offsets using the values from tool 1 in the tool table)

It is an error if:

- the H number is not an integer, or
 - the H number is negative, or
 - the H number is not a valid tool number (though note that 0 is a valid tool number on nonrandom tool changer machines, it means "the tool currently in the spindle")
-

6.3.29 G43.1 Dynamic Tool Length Offset

G43.1 axes

- *G43.1 axes* - change subsequent motions by replacing the current offset(s) of axes. G43.1 does not cause any motion. The next time a compensated axis is moved, that axis's endpoint is the compensated location.

G43.1 Example

```
G90 (set absolute mode)
T1 M6 G43 (load tool 1 and tool length offsets, Z is at machine 0 and DR0 shows Z1.500)
G43.1 Z0.250 (offset current tool offset by 0.250, DR0 now shows Z1.250)
M2 (end program)
```

- See [G90](#) & [T](#) & [M6](#) sections for more information.

It is an error if:

- motion is commanded on the same line as *G43.1*

Note

G43.1 does not write to the tool table.

6.3.30 G43.2 Apply additional Tool Length Offset

G43.2 H- axes-

- *H* - tool number
- *G43.2* - applies an additional simultaneous tool offset.

G43.2 Example

```
G90 (set absolute mode)
T1 M6 (load tool 1)
G43 (or G43 H1 - replace all tool offsets with T1's offset)
G43.2 H10 (also add in T10's tool offset)
M2 (end program)
```

You can sum together an arbitrary number of offsets by calling G43.2 more times. There are no built-in assumptions about which numbers are geometry offsets and which are wear offsets, or that you should have only one of each.

Like the other G43 commands, G43.2 does not cause any motion. The next time a compensated axis is moved, that axis's endpoint is the compensated location.

It is an error if:

- *H* is unspecified and no axis offsets are specified.
- *H* is specified and the given tool number does not exist in the tool table.
- *H* is specified and axes are also specified.

Note

G43.2 does not write to the tool table.

6.3.31 G49 Cancel Tool Length Compensation

- *G49* - cancels tool length compensation

It is OK to program using the same offset already in use. It is also OK to program using no tool length offset if none is currently being used.

6.3.32 G52 Local Coordinate System Offset

G52 axes

G52 is used in a part program as a temporary "local coordinate system offset" within the workpiece coordinate system. More information on G52 is in the [Local and Global Offsets](#) section.

6.3.33 G53 Move in Machine Coordinates

G53 axes

To move in the [machine coordinate system](#), program G53 on the same line as a linear move. *G53* is not modal and must be programmed on each line. *G0* or *G1* does not have to be programmed on the same line if one is currently active.

For example *G53 G0 X0 Y0 Z0* will move the axes to the home position even if the currently selected coordinate system has offsets in effect.

G53 Example

```
G53 G0 X0 Y0 Z0 (rapid linear move to the machine origin)
G53 X2 (rapid linear move to absolute coordinate X2)
```

See [G0](#) section for more information.

It is an error if:

- G53 is used without G0 or G1 being active,
- or G53 is used while cutter compensation is on.

6.3.34 G54-G59.3 Select Coordinate System

- *G54* - select coordinate system 1
 - *G55* - select coordinate system 2
 - *G56* - select coordinate system 3
 - *G57* - select coordinate system 4
 - *G58* - select coordinate system 5
 - *G59* - select coordinate system 6
 - *G59.1* - select coordinate system 7
 - *G59.2* - select coordinate system 8
 - *G59.3* - select coordinate system 9
-

The coordinate systems store the axis values and the XY rotation angle around the Z axis in the parameters shown in the following table.

Table 6.10: Coordinate System Parameters

Select CS		X	Y	Z	A	B	C	U	V	W	R
G54	1	5221	5222	5223	5224	5225	5226	5227	5228	5229	5230
G55	2	5241	5242	5243	5244	5245	5246	5247	5248	5249	5250
G56	3	5261	5262	5263	5264	5265	5266	5267	5268	5269	5270
G57	4	5281	5282	5283	5284	5285	5286	5287	5288	5289	5290
G58	5	5301	5302	5303	5304	5305	5306	5307	5308	5309	5310
G59	6	5321	5322	5323	5324	5325	5326	5327	5328	5329	5330
G59.1	7	5341	5342	5343	5344	5345	5346	5347	5348	5349	5350
G59.2	8	5361	5362	5363	5364	5365	5366	5367	5368	5369	5370
G59.3	9	5381	5382	5383	5384	5385	5386	5387	5388	5389	5390

It is an error if:

- selecting a coordinate system is used while cutter compensation is on.

See the [Coordinate System](#) section for an overview of coordinate systems.

6.3.35 G61 Exact Path Mode

- *G61* - Exact path mode, movement exactly as programmed. Moves will slow or stop as needed to reach every programmed point. If two sequential moves are exactly co-linear movement will not stop.

6.3.36 G61.1 Exact Stop Mode

- *G61.1* - Exact stop mode, movement will stop at the end of each programmed segment.

6.3.37 G64 Path Blending

G64 <P- <Q->>

- *P* - motion blending tolerance
- *Q* - naive cam tolerance
- *G64* - best possible speed. Without *P* means to keep the best speed possible, no matter how far away from the programmed point you end up.
- *G64 P-* - Blend between best speed and deviation tolerance
- *G64 P- <Q- >* blending with tolerance. It is a way to fine tune your system for best compromise between speed and accuracy. The *P-* tolerance means that the actual path will be no more than *P-* away from the programmed endpoint. The velocity will be reduced if needed to maintain the path. In addition, when you activate *G64 P- Q-* it turns on the *Naive CAM Detector*; when there are a series of linear XYZ feed moves at the same [feed rate](#) that are less than *Q-* away from being collinear, they

are collapsed into a single linear move. On G2/G3 moves in the G17 (XY) plane when the maximum deviation of an arc from a straight line is less than the G64 P- tolerance the arc is broken into two lines (from start of arc to midpoint, and from midpoint to end). those lines are then subject to the naive cam algorithm for lines. Thus, line-arc, arc-arc, and arc-line cases as well as line-line benefit from the *naive cam detector*. This improves contouring performance by simplifying the path. It is OK to program for the mode that is already active. See also the [Trajectory Control](#) section for more information on these modes. If Q is not specified then it will have the same behavior as before and use the value of P-.

G64 P- Example Line

```
G64 P0.015 (set path following to be within 0.015 of the actual path)
```

It is a good idea to include a path control specification in the preamble of each G-code file.

6.3.38 G70 Lathe finishing cycle

```
G70 Q- <X-> <Z-> <D-> <E-> <P->
```

- *Q* - The subroutine number.
- *X* - The starting X position, defaults to the initial position.
- *Z* - The starting Z position, defaults to the initial position.
- *D* - The starting distance of the profile, defaults to 0.
- *E* - The ending distance of the profile, defaults to 0.
- *P* - The number of passes to use, defaults to 1.

The *G70* cycle is intended to be used after the shape of the profile given in the subroutine with number *Q* has been cut with *G71* or *G72*.

- Preliminary motion.
 - If *Z* or *X* are used a [rapid move](#) to that position is done. This position is also used between each finishing pass.
 - Then a [rapid move](#) to the start of the profile is executed.
 - The path given in *Q*- is followed using the [G1](#) and Section [6.3.5](#) commands.
 - If a next pass is required there is another rapid to the intermediate location, before a rapid is done to the start of the profile.
 - After the final pass, the tool is left at the end of the profile including *E*-.
- Multiple passes. The distance between the pass and the final profile is $(\text{pass}-1) \cdot (D-E) / P + E$. Where pass the pass number and *D*, *E* and *P* are the *D/E/P* numbers.
- The distance is computed using the starting position of the cycle, with a positive distance towards this point.
- Fillet and chamfers in the profile. It is possible to add fillets or chamfers in the profile, see Section [6.3.39](#) for more details.

It is an error if:

- There is no subroutine defined with the number given in *Q*.
- The path given in the profile is not monotonic in *Z* or *X*.
- Section [6.3.18](#) has not been used to select the ZX plane.

6.3.39 G71 G72 Lathe roughing cycles

```
G71 Q- <X-> <Z-> <D-> <I-> <R->
G71.1 Q- <X-> <Z-> <D-> <I-> <R->
G71.2 Q- <X-> <Z-> <D-> <I-> <R->
G72 Q- <X-> <Z-> <D-> <I-> <R->
G72.1 Q- <X-> <Z-> <D-> <I-> <R->
G72.2 Q- <X-> <Z-> <D-> <I-> <R->
```

- *Q* - The subroutine number.
- *X* - The starting X position, defaults to the initial position.
- *Z* - The starting Z position, defaults to the initial position.
- *D* - The remaining distance to the profile, defaults to 0.
- *I* - The cutting increment, defaults to 1.
- *R* - The retracting distance, defaults to 0.5.

The G71/G72 cycle is intended to rough cut a profile on a lathe. The G71 cycles remove layers of the material while traversing in the Z direction. The G72 cycles remove material while traversing the X axis, the so called facing cycle. The direction of travel is the same as in the path given in the subroutine. For the G71 cycle the Z coordinate must be monotonically changing, for the G72 this is required for the X axis.

The profile is given in a subroutine with number Q-. This subroutine may contain G0, G1, G2 and G3 motion commands. All other commands are ignored, including feed and speed settings. The Section 6.3.3 commands are interpreted as G1 commands. Each motion command may also include an optional A- or C- number. If the number A- is added a fillet with the radius given by A will be inserted at the endpoint of that motion, if this radius is too large the algorithm will fail with a non-monotonic path error. It is also possible to use the C- number, which allows a chamfer to be inserted. This chamfer has the same endpoints as a fillet of the same dimension would have but a straight line is inserted instead of an arc.

When in absolute mode the U (for X) and W (for Z) can be used as incremental displacements.

The G7x.1 cycles do not cut pockets. The G7x.2 cycles only cut after the first pocket and continue where G7x.1 stopped. It is advisable to leave some additional material to cut before the G7x.2 cycle, so if G7x.1 used a D1.0 the G7x.2 can use D0.5 and 0.5mm will be removed while moving from one pocket to the next.

The normal G7x cycles cut the entire profile in one cycle.

1. Preliminary motion.

- If Z or X are used a **rapid move** to that position is done.
- After the profile has been cut, the tool stops at the end of the profile, including the distance specified in D.

2. The D number is used to keep a distance from the final profile, to allow material to remain for finishing.

It is an error if:

- There is no subroutine defined with the number given in Q.
- The path given in the profile is not monotonic in Z or X.
- Section 6.3.18 has not been used to select the ZX plane.
- Section 6.3.26 is active.

6.3.40 G73 Drilling Cycle with Chip Breaking

G73 X- Y- Z- R- Q- <L->

- *R* - retract position along the Z axis.
- *Q* - delta increment along the Z axis.
- *L* - repeat

The *G73* cycle is drilling or milling with chip breaking. This cycle takes a *Q* number which represents a *delta* increment along the Z axis.

- Preliminary motion.
 - If the current Z position is below the R position, The Z axis does a [rapid move](#) to the R position.
 - Move to the X Y coordinates
- Move the Z-axis only at the current [feed rate](#) downward by delta or to the Z position, whichever is less deep.
- Rapid up .010 of an inch or 0.254 mm.
- Repeat steps 2 and 3 until the Z position is reached at step 2.
- The Z axis does a rapid move to the R position.

It is an error if:

- the *Q* number is negative or zero.
- the *R* number is not specified

6.3.41 G74 Left-hand Tapping Cycle with Dwell

G74 (X- Y- Z-) or (U- V- W-) R- L- P- \$- F-

- *R* - Retract position along the Z axis.
- *L* - Used in incremental mode; number of times to repeat the cycle. See [G81](#) for examples.
- *P* - Dwell time (seconds).
- *\$* - Selected spindle.
- *F* - Feed rate (spindle speed multiplied by distance traveled per revolution (thread pitch)).



Warning

G74 does not use synchronized motion.

The *G74* cycle is intended for tapping with floating chuck and dwell at the bottom of the hole.

1. Preliminary motion, as described in the [Preliminary and In-Between Motion](#) section.
-

2. Disable Feed and Speed Overrides.
3. Move the Z-axis at the current feed rate to the Z position.
4. Stop the selected spindle (chosen by the \$ parameter)
5. Start spindle rotation clockwise.
6. Dwell for the P number of seconds.
7. Move the Z-axis at the current feed rate to clear Z
8. Restore Feed and Speed override enables to previous state

The length of the dwell is specified by a *P*- word in the G74 block. The feed rate *F*- is spindle speed multiplied by distance per revolution (thread pitch). In example S100 with 1.25MM per revolution thread pitch gives a feed of F125.

6.3.42 G76 Threading Cycle

G76 P- Z- I- J- R- K- Q- H- E- L- \$-

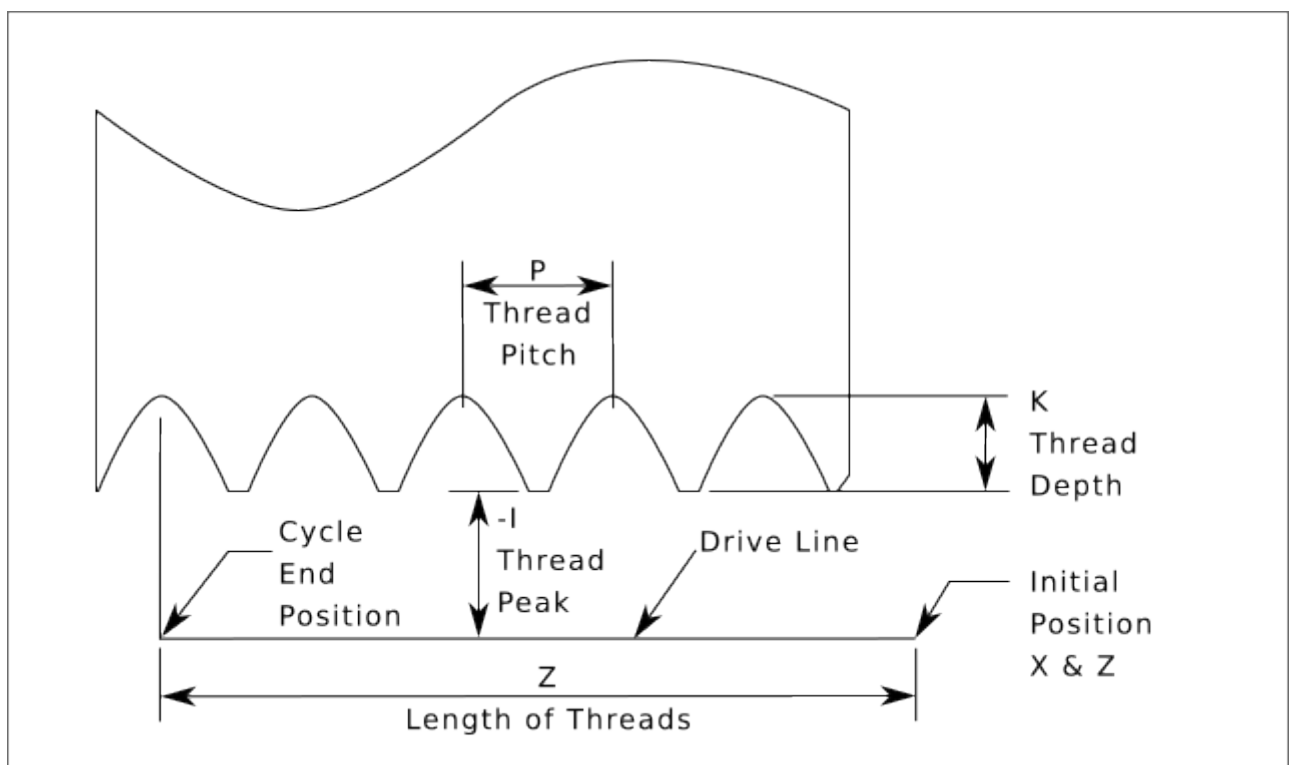


Figure 6.6: G76 Threading

- *Drive Line* - A line through the initial X position parallel to the Z.
- *P* - The *thread pitch* in distance per revolution.
- *Z* - The final position of threads. At the end of the cycle the tool will be at this Z position.

Note

When G7 *Lathe Diameter Mode* is in force the values for *I*, *J* and *K* are diameter measurements. When G8 *Lathe Radius Mode* is in force the values for *I*, *J* and *K* are radius measurements.

- *I* - The *thread peak* offset from the *drive line*. Negative *I* values are external threads, and positive *I* values are internal threads. Generally the material has been turned to this size before the G76 cycle.
- *J* - A positive value specifying the *initial cut depth*. The first threading cut will be *J* beyond the *thread peak* position.
- *K* - A positive value specifying the *full thread depth*. The final threading cut will be *K* beyond the *thread peak* position.

Optional settings

- *\$* - The spindle number to which the motion will be synchronised (default 0). For example is \$1 is programmed then the motion will begin on the reset of spindle.1.index-enable and proceed in synchrony with the value of spindle.1.revs
- *R* - The *depth degression*. *R1.0* selects constant depth on successive threading passes. *R2.0* selects constant area. Values between 1.0 and 2.0 select decreasing depth but increasing area. Values above 2.0 select decreasing area. Beware that unnecessarily high degression values will cause a large number of passes to be used. (degression = a descent by stages or steps.)

**Warning**

Unnecessarily high degression values will produce an unnecessarily high number of passes. (degression = dive in stages)

- *Q* - The *compound slide angle* is the angle (in degrees) describing to what extent successive passes should be offset along the drive line. This is used to cause one side of the tool to remove more material than the other. A positive *Q* value causes the leading edge of the tool to cut more heavily. Typical values are 29, 29.5 or 30.
- *H* - The number of *spring passes*. Spring passes are additional passes at full thread depth. If no additional passes are desired, program *H0*.

Thread entries and exits can be programmed tapered with the *E* and *L* values.

- *E* - Specifies the distance along the drive line used for the taper. The angle of the taper will be so the last pass tapers to the thread crest over the distance specified with *E*. *E0.2* will give a taper for the first/last 0.2 length units along the thread. For a 45 degree taper program *E* the same as *K*.
- *L* - Specifies which ends of the thread get the taper. Program *L0* for no taper (the default), *L1* for entry taper, *L2* for exit taper, or *L3* for both entry and exit tapers. Entry tapers will pause at the drive line to synchronize with the index pulse then move at the [feed rate](#) in to the beginning of the taper. No entry taper and the tool will rapid to the cut depth then synchronize and begin the cut.

The tool is moved to the initial X and Z positions prior to issuing the G76. The X position is the *drive line* and the Z position is the start of the threads.

The tool will pause briefly for synchronization before each threading pass, so a relief groove will be required at the entry unless the beginning of the thread is past the end of the material or an entry taper is used.

Unless using an exit taper, the exit move is not synchronized to the spindle speed and will be a [rapid move](#). With a slow spindle, the exit move might take only a small fraction of a revolution. If the spindle speed is increased after several passes are complete, subsequent exit moves will require a larger portion of a revolution, resulting in a very heavy cut during the exit move. This can be avoided by providing a relief groove at the exit, or by not changing the spindle speed while threading.

The final position of the tool will be at the end of the *drive line*. A safe Z move will be needed with an internal thread to remove the tool from the hole.

It is an error if:

- The active plane is not the ZX plane.
- Other axis words, such as X- or Y-, are specified.
- The *R*- degression value is less than 1.0.
- All the required words are not specified.
- *P*-, *J*-, *K*- or *H*- is negative.
- *E*- is greater than half the drive line length.

HAL Connections The pins *spindle.N.at-speed* and the *encoder.n.phase-Z* for the spindle must be connected in your HAL file before G76 will work. See the [spindle](#) pins in the Motion section for more information.

Technical Info The G76 canned cycle is based on the G33 Spindle Synchronized Motion. For more information see the G33 [Technical Info](#).

The sample program *g76.ngc* shows the use of the G76 canned cycle, and can be previewed and executed on any machine using the *sim/lathe.ini* configuration.

G76 Example Code

```
G0 Z-0.5 X0.2
G76 P0.05 Z-1 I-.075 J0.008 K0.045 Q29.5 L2 E0.045
```

In the figure the tool is in the final position after the G76 cycle is completed. You can see the entry path on the right from the Q29.5 and the exit path on the left from the L2 E0.045. The white lines are the cutting moves.

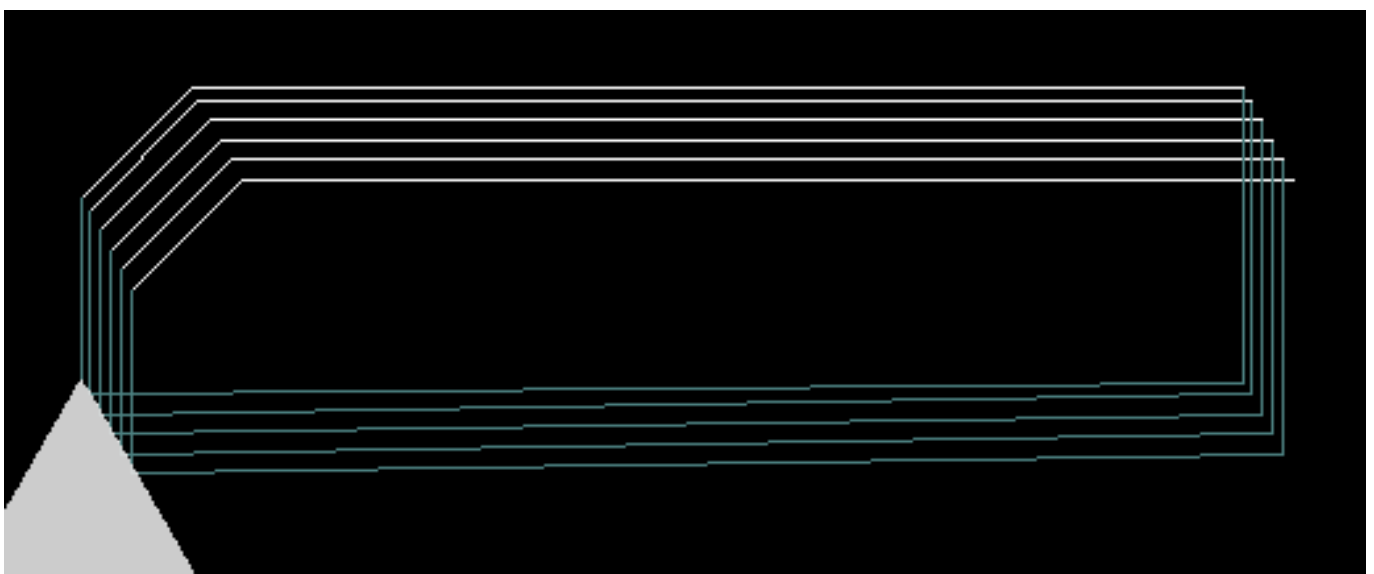


Figure 6.7: G76 Example

6.3.43 G80-G89 Canned Cycles

The canned cycles *G81* through *G89* and the canned cycle stop *G80* are described in this section.

All canned cycles are performed with respect to the currently-selected plane. Any of the nine planes may be selected. Throughout this section, most of the descriptions assume the XY-plane has been selected. The behavior is analogous if another plane is selected, and the correct words must be used. For instance, in the *G17.1* plane, the action of the canned cycle is along W, and the locations or increments are given with U and V. In this case substitute U,V,W for X,Y,Z in the instructions below.

Rotary axis words are not allowed in canned cycles. When the active plane is one of the XYZ family, the UVW axis words are not allowed. Likewise, when the active plane is one of the UVW family, the XYZ axis words are not allowed.

6.3.43.1 Common Words

All canned cycles use X, Y, Z, or U, V, W groups depending on the plane selected and R words. The R (usually meaning retract) position is along the axis perpendicular to the currently selected plane (Z-axis for XY-plane, etc.) Some canned cycles use additional arguments.

6.3.43.2 Sticky Words

For canned cycles, we will call a number *sticky* if, when the same cycle is used on several lines of code in a row, the number must be used the first time, but is optional on the rest of the lines. Sticky numbers keep their value on the rest of the lines if they are not explicitly programmed to be different. The R number is always sticky.

In incremental distance mode X, Y, and R numbers are treated as increments from the current position and Z as an increment from the Z-axis position before the move involving Z takes place. In absolute distance mode, the X, Y, R, and Z numbers are absolute positions in the current coordinate system.

6.3.43.3 Repeat Cycle

The L number is optional and represents the number of repeats. L=0 is not allowed. If the repeat feature is used, it is normally used in incremental distance mode, so that the same sequence of motions is repeated in several equally spaced places along a straight line. When L- is greater than 1 in incremental mode with the XY-plane selected, the X and Y positions are determined by adding the given X and Y numbers either to the current X and Y positions (on the first go-around) or to the X and Y positions at the end of the previous go-around (on the repetitions). Thus, if you program *L10*, you will get 10 cycles. The first cycle will be distance X,Y from the original location. The R and Z positions do not change during the repeats. The L number is not sticky. In absolute distance mode, L>1 means *do the same cycle in the same place several times*, Omitting the L word is equivalent to specifying L=1.

6.3.43.4 Retract Mode

The height of the retract move at the end of each repeat (called *clear Z* in the descriptions below) is determined by the setting of the retract mode, either to the original Z position (if that is above the R position and the retract mode is *G98*, OLD_Z), or otherwise to the R position. See the [G98 G99](#) section.

6.3.43.5 Canned Cycle Errors

It is an error if:

- axis words are all missing during a canned cycle,
- axis words from different groups (XYZ) (UVW) are used together,
- a P number is required and a negative P number is used,
- an L number is used that does not evaluate to a positive integer,
- rotary axis motion is used during a canned cycle,
- inverse time feed rate is active during a canned cycle,
- or cutter compensation is active during a canned cycle.

If the XY plane is active, the Z number is sticky, and it is an error if:

- the Z number is missing and the same canned cycle was not already active,
- or the R number is less than the Z number.

If other planes are active, the error conditions are analogous to the XY conditions above.

6.3.43.6 Preliminary and In-Between Motion

Preliminary motion is a set of motions that is common to all of the milling canned cycles. If the current Z position is below the R position, the Z axis does a [rapid move](#) to the R position. This happens only once, regardless of the value of L.

In addition, at the beginning of the first cycle and each repeat, the following one or two moves are made:

- A [rapid move](#) parallel to the XY-plane to the given XY-position.
- The Z-axis make a rapid move to the R position, if it is not already at the R position.

If another plane is active, the preliminary and in-between motions are analogous.

6.3.43.7 Why use a canned cycle?

There are at least two reasons for using canned cycles. The first is the economy of code. A single bore would take several lines of code to execute.

The G81 [Example 1](#) demonstrates how a canned cycle could be used to produce 8 holes with ten lines of G-code within the canned cycle mode. The program below will produce the same set of 8 holes using five lines for the canned cycle. It does not follow exactly the same path nor does it drill in the same order as the earlier example. But the program writing economy of a good canned cycle should be obvious.

Note

Line numbers are not needed but help clarify these examples.

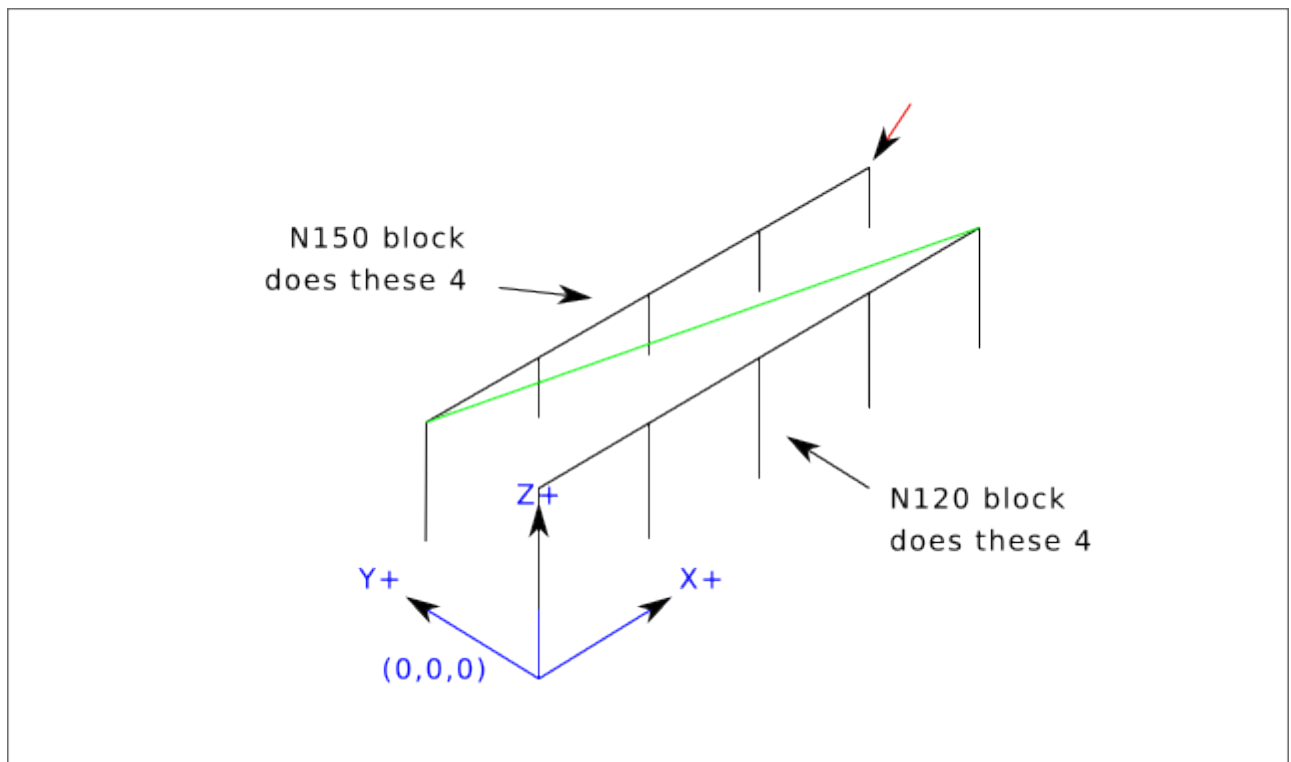
Eight Holes

```

N100 G90 G0 X0 Y0 Z0 (move coordinate home)
N110 G1 F10 X0 G4 P0.1
N120 G91 G81 X1 Y0 Z-1 R1 L4(canned drill cycle)
N130 G90 G0 X0 Y1
N140 Z0
N150 G91 G81 X1 Y0 Z-0.5 R1 L4(canned drill cycle)
N160 G80 (turn off canned cycle)
N170 M2 (program end)

```

The G98 on the second line above means that the return move will be to the Z value on the first line since it is higher than the specified R value.

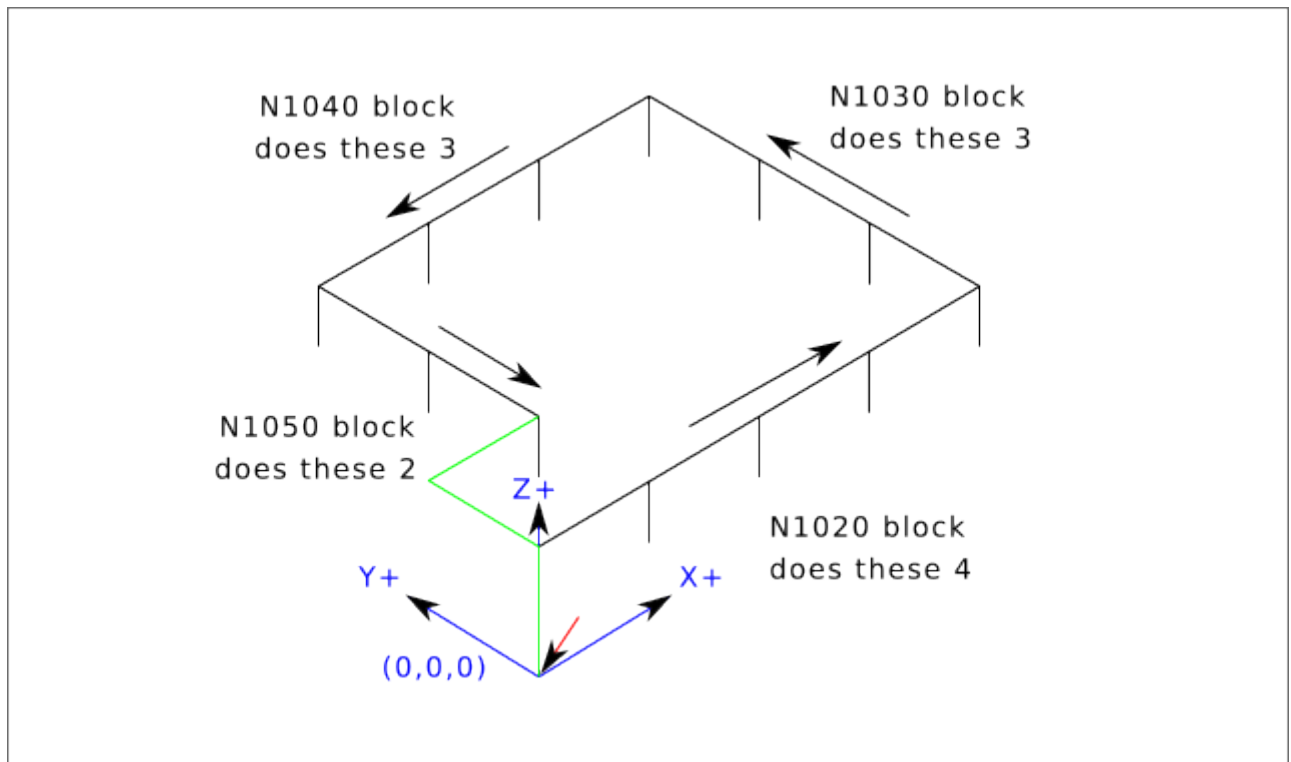


Twelve Holes in a Square This example demonstrates the use of the L word to repeat a set of incremental drill cycles for successive blocks of code within the same G81 motion mode. Here we produce 12 holes using five lines of code in the canned motion mode.

```

N1000 G90 G0 X0 Y0 Z0 (move coordinate home)
N1010 G1 F50 X0 G4 P0.1
N1020 G91 G81 X1 Y0 Z-0.5 R1 L4 (canned drill cycle)
N1030 X0 Y1 R0 L3 (repeat)
N1040 X-1 Y0 L3 (repeat)
N1050 X0 Y-1 L2 (repeat)
N1060 G80 (turn off canned cycle)
N1070 G90 G0 X0 (rapid move home)
N1080 Y0
N1090 Z0
N1100 M2 (program end)

```



The second reason to use a canned cycle is that they all produce preliminary moves and returns that you can anticipate and control regardless of the start point of the canned cycle.

6.3.44 G80 Cancel Canned Cycle

- *G80* - cancel canned cycle modal motion. *G80* is part of modal group 1, so programming any other G-code from modal group 1 will also cancel the canned cycle.

It is an error if:

- Axis words are programmed when *G80* is active.

G80 Example

```
G90 G81 X1 Y1 Z1.5 R2.8 (absolute distance canned cycle)
G80 (turn off canned cycle motion)
G0 X0 Y0 Z0 (rapid move to coordinate home)
```

The following code produces the same final position and machine state as the previous code.

G0 Example

```
G90 G81 X1 Y1 Z1.5 R2.8 (absolute distance canned cycle)
G0 X0 Y0 Z0 (rapid move to coordinate home)
```

The advantage of the first set is that, the *G80* line clearly turns off the *G81* canned cycle. With the first set of blocks, the programmer must turn motion back on with *G0*, as is done in the next line, or any other motion mode G word.

If a canned cycle is not turned off with *G80* or another motion word, the canned cycle will attempt to repeat itself using the next block of code that contains an X, Y, or Z word. The following file drills (*G81*) a set of eight holes as shown in the following caption.

G80 Example 1

```
N100 G90 G0 X0 Y0 Z0 (coordinate home)
N110 G1 X0 G4 P0.1
N120 G81 X1 Y0 Z0 R1 (canned drill cycle)
N130 X2
N140 X3
N150 X4
N160 Y1 Z0.5
N170 X3
N180 X2
N190 X1
N200 G80 (turn off canned cycle)
N210 G0 X0 (rapid move home)
N220 Y0
N230 Z0
N240 M2 (program end)
```

Note

Notice the Z position change after the first four holes. Also, this is one of the few places where line numbers have some value, being able to point a reader to a specific line of code.

The use of G80 in line N200 is optional because the G0 on the next line will turn off the G81 cycle. But using the G80 as shown in Example 1, will provide for easier to read canned cycle. Without it, it is not so obvious that all of the blocks between N120 and N200 belong to the canned cycle.

6.3.45 G81 Drilling Cycle

G81 (X- Y- Z-) or (U- V- W-) R- L-

The *G81* cycle is intended for drilling.

The cycle functions as follows:

- Preliminary motion, as described in the [Preliminary and In-Between Motion](#) section.
- Move the Z-axis at the current [feed rate](#) to the Z position.
- The Z-axis does a [rapid move](#) to clear Z.

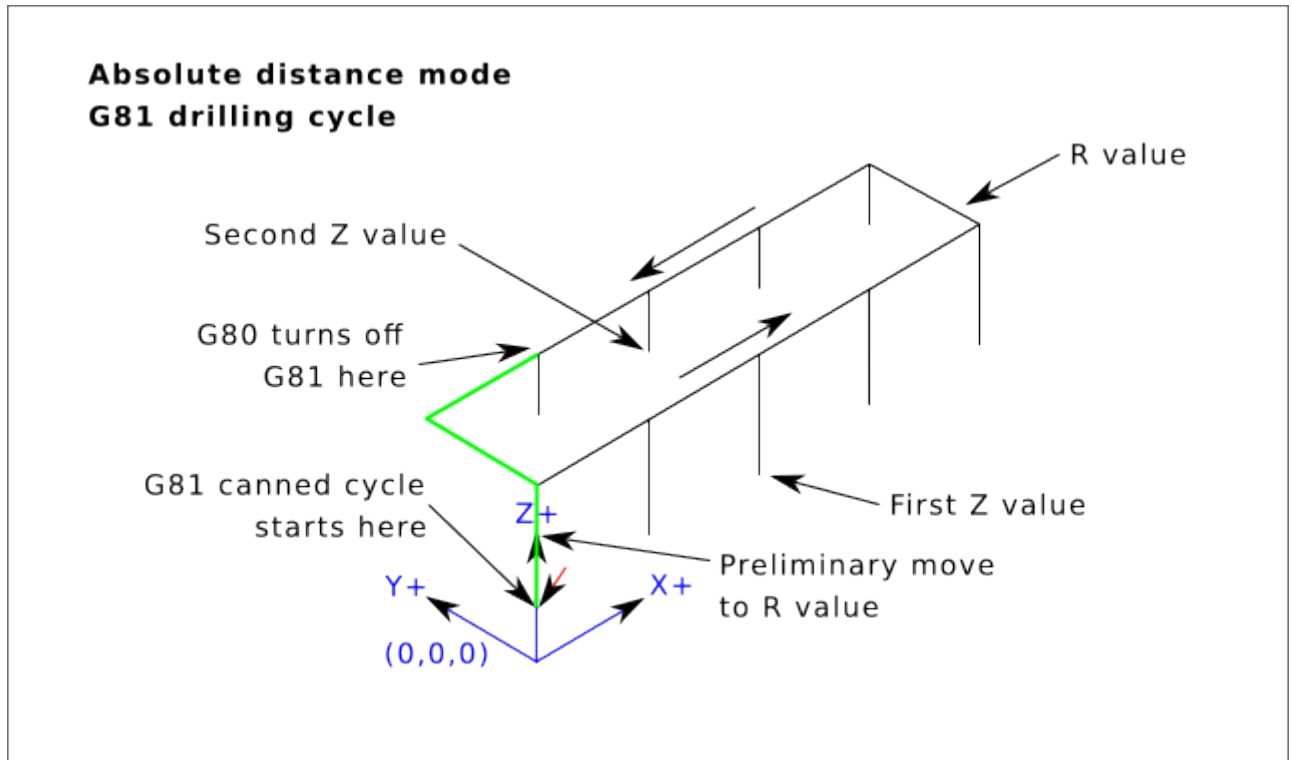


Figure 6.8: G81 Cycle

Example 1 - Absolute Position G81

```
G90 G98 G81 X4 Y5 Z1.5 R2.8
```

Suppose the current position is (X1, Y2, Z3) and the preceding line of NC code is interpreted.

This calls for absolute distance mode (G90) and OLD_Z retract mode (G98) and calls for the G81 drilling cycle to be performed once.

- The X value and X position are 4.
- The Y value and Y position are 5.
- The Z value and Z position are 1.5.
- The R value and clear Z are 2.8. OLD_Z is 3.

The following moves take place:

- A **rapid move** parallel to the XY plane to (X4, Y5)
- A rapid move move parallel to the Z-axis to (Z2.8).
- Move parallel to the Z-axis at the **feed rate** to (Z1.5)
- A rapid move parallel to the Z-axis to (Z3)



Example 2 - Relative Position G81

```
G91 G98 G81 X4 Y5 Z-0.6 R1.8 L3
```

Suppose the current position is (X1, Y2, Z3) and the preceding line of NC code is interpreted.

This calls for incremental distance mode (G91) and OLD_Z retract mode (G98). It also calls for the G81 drilling cycle to be repeated three times. The X value is 4, the Y value is 5, the Z value is -0.6 and the R value is 1.8. The initial X position is 5 ($=1+4$), the initial Y position is 7 ($=2+5$), the clear Z position is 4.8 ($=1.8+3$), and the Z position is 4.2 ($=4.8-0.6$). OLD_Z is 3.

The first preliminary move is a maximum rapid move along the Z axis to (X1,Y2,Z4.8), since OLD_Z < clear Z.

The first repeat consists of 3 moves.

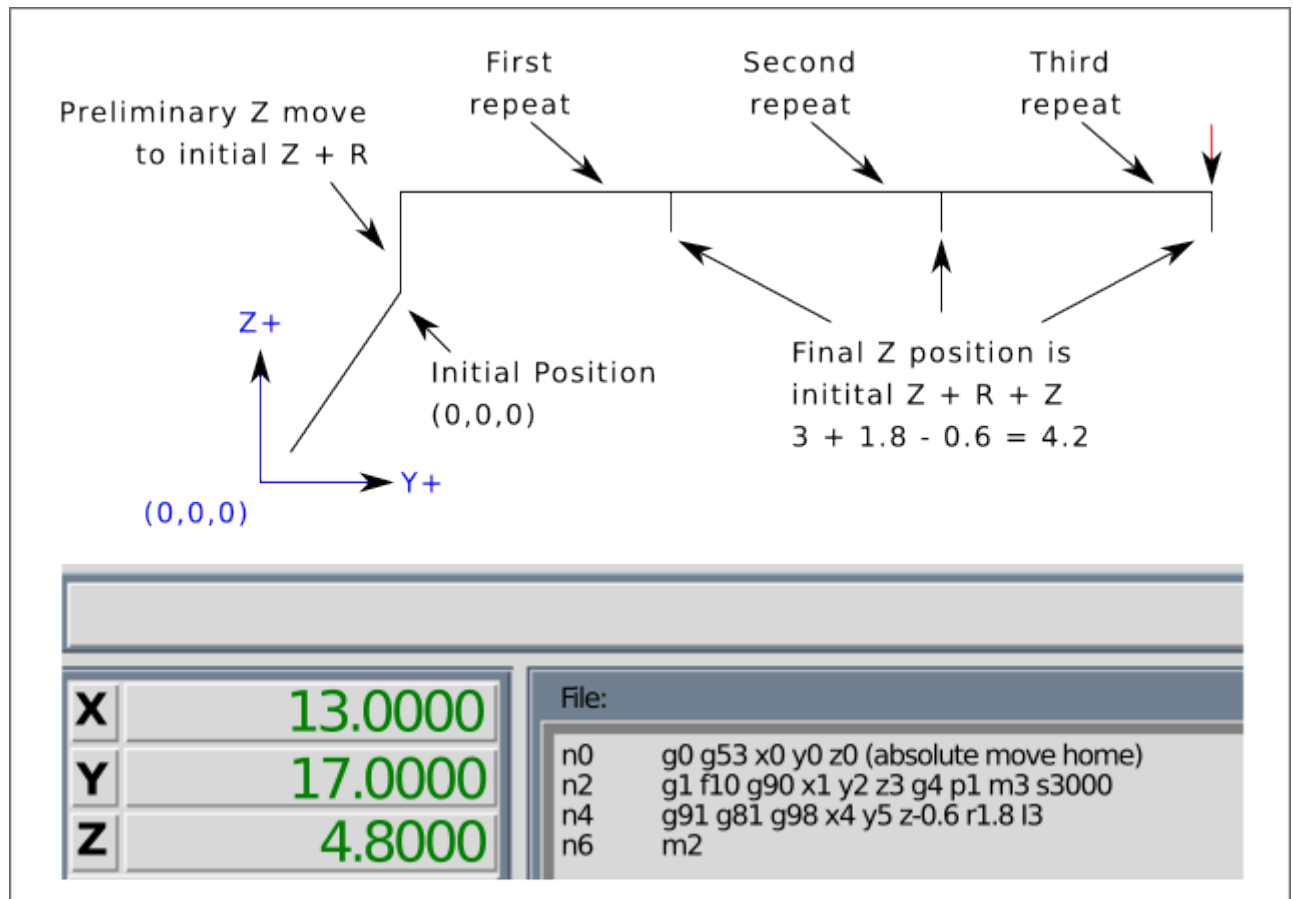
- A **rapid move** parallel to the XY-plane to (X5, Y7)
- Move parallel to the Z-axis at the **feed rate** to (Z4.2)
- A rapid move parallel to the Z-axis to (X5, Y7, Z4.8)

The second repeat consists of 3 moves. The X position is reset to 9 ($=5+4$) and the Y position to 12 ($=7+5$).

- A **rapid move** parallel to the XY-plane to (X9, Y12, Z4.8)
- Move parallel to the Z-axis at the feed rate to (X9, Y12, Z4.2)
- A rapid move parallel to the Z-axis to (X9, Y12, Z4.8)

The third repeat consists of 3 moves. The X position is reset to 13 ($=9+4$) and the Y position to 17 ($=12+5$).

- A **rapid move** parallel to the XY-plane to (X13, Y17, Z4.8)
- Move parallel to the Z-axis at the feed rate to (X13, Y17, Z4.2)
- A rapid move parallel to the Z-axis to (X13, Y17, Z4.8)

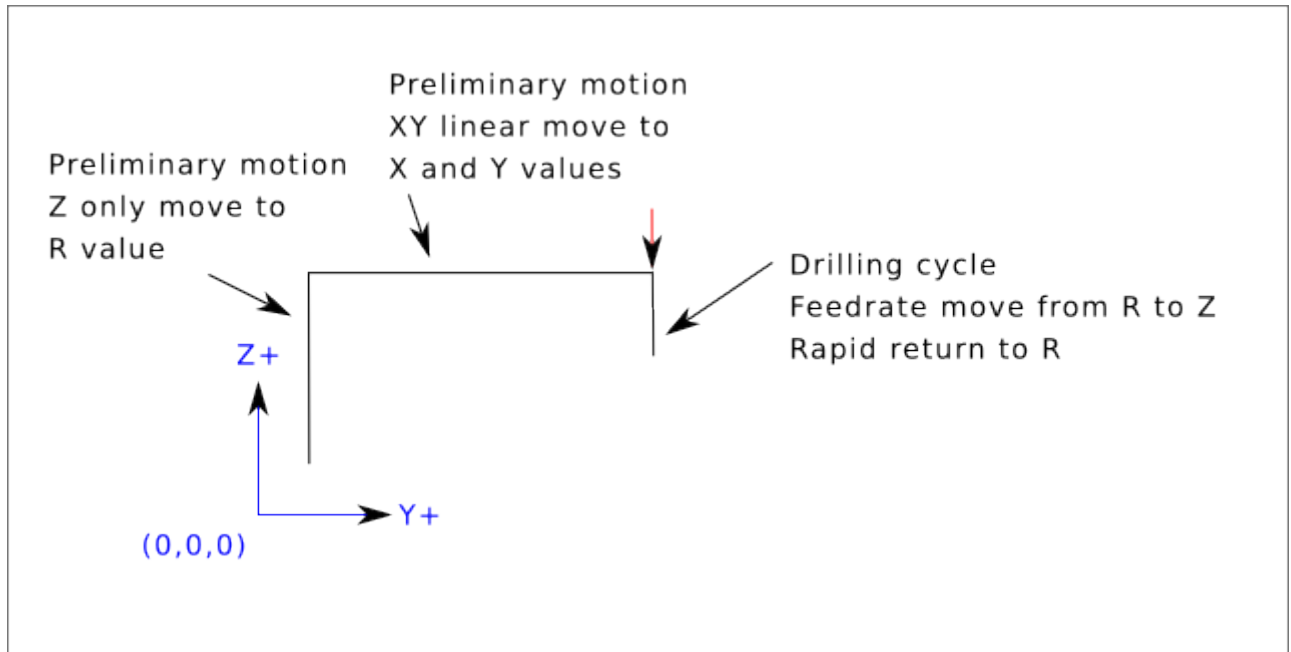


Example 3 - Relative Position G81

G90 G98 G81 X4 Y5 Z1.5 R2.8

Now suppose that you execute the first G81 block of code but from (X0, Y0, Z0) rather than from (X1, Y2, Z3).

Since OLD_Z is below the R value, it adds nothing for the motion but since the initial value of Z is less than the value specified in R, there will be an initial Z move during the preliminary moves.



Example 4 - Absolute G81 R > Z This is a plot of the path of motion for the second g81 block of code.

```
G91 G98 G81 X4 Y5 Z-0.6 R1.8 L3
```

Since this plot starts with (X0, Y0, Z0), the interpreter adds the initial Z0 and R1.8 and rapid moves to that location. After that initial Z move, the repeat feature works the same as it did in example 3 with the final Z depth being 0.6 below the R value.



Example 5 - Relative position R > Z

```
G90 G98 G81 X4 Y5 Z-0.6 R1.8
```

Since this plot starts with (X0, Y0, Z0), the interpreter adds the initial Z0 and R1.8 and rapid moves to that location as in *Example 4*. After that initial Z move, the **rapid move** to X4 Y5 is done. Then the

final Z depth being 0.6 below the R value. The repeat function would make the Z move in the same location again.

6.3.46 G82 Drilling Cycle, Dwell

```
G82 (X- Y- Z-) or (U- V- W-) R- L- P-
```

The G82 cycle is intended for drilling with a dwell at the bottom of the hole.

- Preliminary motion, as described in the [Preliminary and In-Between Motion](#) section.
- Move the Z-axis at the current [feed rate](#) to the Z position.
- Dwell for the P number of seconds.
- The Z-axis does a [rapid move](#) to clear Z.

The motion of a G82 canned cycle looks just like G81 with the addition of a dwell at the bottom of the Z move. The length of the dwell is specified by a P- word in the G82 block.

```
G90 G82 G98 X4 Y5 Z1.5 R2.8 P2
```

This will be similar to example 3 above, just with an added dwell of 2 seconds at the bottom of the hole.

6.3.47 G83 Peck Drilling Cycle

```
G83 (X- Y- Z-) or (U- V- W-) R- L- Q-
```

The G83 cycle (often called peck drilling) is intended for deep drilling or milling with chip breaking. The retracts in this cycle clear the hole of chips and cut off any long stringers (which are common when drilling in aluminum). This cycle takes a Q number which represents a *delta* increment along the Z-axis. The retract before final depth will always be to the *retract* plane even if G98 is in effect. The final retract will honor the G98/99 in effect. G83 functions the same as G81 with the addition of retracts during the drilling operation.

- Preliminary motion, as described in the [Preliminary and In-Between Motion](#) section.
- Move the Z-axis at the current [feed rate](#) downward by delta or to the Z position, whichever is less deep.
- Rapid move back out to the retract plane specified by the R word.
- Rapid move back down to the current hole bottom, less .010 of an inch or 0.254 mm.
- Repeat steps 2, 3, and 4 until the Z position is reached at step 2.
- The Z-axis does a [rapid move](#) to clear Z.

It is an error if:

- the Q number is negative or zero.
-

6.3.48 G84 Right-hand Tapping Cycle, Dwell

G84 (X- Y- Z-) or (U- V- W-) R- L- P- \$- F-

- *R* - Retract position along the Z axis.
- *L* - Used in incremental mode; number of times to repeat the cycle. See [G81](#) for examples.
- *P* - Dwell time (seconds).
- *\$* - Selected spindle.
- *F* - Feed rate (spindle speed multiplied by distance traveled per revolution (thread pitch)).



Warning

G84 does not use synchronized motion.

The *G84* cycle is intended for tapping with floating chuck and dwell at the bottom of the hole.

- Preliminary motion, as described in the [Preliminary and In-Between Motion](#) section.
- Disable Feed and Speed Overrides.
- Move the Z-axis at the current feed rate to the Z position.
- Stop the selected spindle (chosen by the \$ parameter)
- Start spindle rotation counterclockwise.
- Dwell for the P number of seconds.
- Move the Z-axis at the current feed rate to clear Z
- Restore Feed and Speed override enables to previous state

The length of the dwell is specified by a *P*- word in the G84 block. The feed rate *F*- is spindle speed multiplied by distance per revolution (thread pitch). In example S100 with 1.25MM per revolution thread pitch gives a feed of F125.

6.3.49 G85 Boring Cycle, Feed Out

G85 (X- Y- Z-) or (U- V- W-) R- L-

The *G85* cycle is intended for boring or reaming, but could be used for drilling or milling.

- Preliminary motion, as described in the [Preliminary and In-Between Motion](#) section.
- Move the Z-axis only at the current [feed rate](#) to the Z position.
- Retract the Z-axis at the current feed rate to the R plane if it is lower than the initial Z.
- Retract at the traverse rate to clear Z.

6.3.50 G86 Boring Cycle, Spindle Stop, Rapid Move Out

G86 (X- Y- Z-) or (U- V- W-) R- L- P- \$-

The *G86* cycle is intended for boring. This cycle uses a P number for the number of seconds to dwell.

- Preliminary motion, as described in the [Preliminary and In-Between Motion](#) section.
- Move the Z-axis only at the current [feed rate](#) to the Z position.
- Dwell for the P number of seconds.
- Stop the selected spindle turning. (Chosen by the \$ parameter)
- The Z-axis does a [rapid move](#) to clear Z.
- Restart the spindle in the direction it was going.

It is an error if:

- the spindle is not turning before this cycle is executed.

6.3.51 G87 Back Boring Cycle

This code is currently unimplemented in LinuxCNC. It is accepted, but the behavior is undefined.

6.3.52 G88 Boring Cycle, Spindle Stop, Manual Out

This code is currently unimplemented in LinuxCNC. It is accepted, but the behavior is undefined.

6.3.53 G89 Boring Cycle, Dwell, Feed Out

G89 (X- Y- Z-) or (U- V- W-) R- L- P-

The *G89* cycle is intended for boring. This cycle uses a P number, where P specifies the number of seconds to dwell.

- Preliminary motion, as described in the [Preliminary and In-Between Motion](#) section.
- Move the Z-axis only at the current [feed rate](#) to the Z position.
- Dwell for the P number of seconds.
- Retract the Z-axis at the current feed rate to clear Z.

6.3.54 G90, G91 Distance Mode

- *G90* - absolute distance mode In absolute distance mode, axis numbers (X, Y, Z, A, B, C, U, V, W) usually represent positions in terms of the currently active coordinate system. Any exceptions to that rule are described explicitly in the [G80 G89](#) section.
 - *G91* - incremental distance mode In incremental distance mode, axis numbers usually represent increments from the current coordinate.
-

G90 Example

G90 (set absolute distance mode)
 G0 X2.5 (rapid move to coordinate X2.5 including any offsets in effect)

G91 Example

G91 (set incremental distance mode)
 G0 X2.5 (rapid move 2.5 from current position along the X axis)

- See [G0](#) section for more information.

6.3.55 G90.1, G91.1 Arc Distance Mode

- *G90.1* - absolute distance mode for I, J & K offsets. When G90.1 is in effect I and J both must be specified with G2/3 for the XY plane or J and K for the XZ plane or it is an error.
- *G91.1* - incremental distance mode for I, J & K offsets. G91.1 Returns I, J & K to their default behavior.

6.3.56 G92 Coordinate System Offset

G92 axes



Warning

Only use G92 after your machine has been positioned to the desired point.

G92 makes the current point have the coordinates you want (without motion), where the axis words contain the axis numbers you want. All axis words are optional, except that at least one must be used. If an axis word is not used for a given axis, the offset for that axis will be zero.

When G92 is executed, the [origins](#) of all coordinate systems move. They move such that the value of the current controlled point, in the currently active coordinate system, becomes the specified value. All of the coordinate system's origins (G53-G59.3) are offset this same distance.

G92 uses the values stored in [parameters](#) 5211-5219 as the X Y Z A B C U V W offset values for each axis. The parameter values are *absolute* machine coordinates in the native machine *units* as specified in the INI file. All axes defined in the INI file will be offset when G92 is active. If an axis was not entered following the G92, that axis' offset will be zero.

For example, suppose the current point is at X=4 and there is currently no G92 offset active. Then G92 X7 is programmed. This moves all origins -3 in X, which causes the current point to become X=7. This -3 is saved in parameter 5211.

Being in incremental distance mode (G91 instead of G90) has no effect on the action of G92.

G92 offsets may be already be in effect when the G92 is called. If this is the case, the offset is replaced with a new offset that makes the current point become the specified value.

It is an error if all axis words are omitted.

LinuxCNC stores the G92 offsets and reuses them on the next run of a program. To prevent this, one can program a G92.1 (to erase them), or program a G92.2 (to remove them - they are still stored).

Note

The *G52* command can also be used to change this offset; see the [Offsets](#) section for more details about *G92* and *G52* and how they interact.

See the [Coordinate System](#) section for an overview of coordinate systems.

See the [Parameters](#) section for more information.

6.3.57 G92.1, G92.2 Reset G92 Offsets

- *G92.1* - turn off G92 offsets and reset [parameters](#) 5211 - 5219 to zero.
- *G92.2* - turn off G92 offsets but keep [parameters](#) 5211 - 5219 available.

Note

G92.1 only clears G92 offsets, to change G53-G59.3 coordinate system offsets in G-code use either [G10 L2](#) or [G10 L20](#).

6.3.58 G92.3 Restore G92 Offsets

- *G92.3* - set the G92 offset to the values saved in parameters 5211 to 5219

You can set axis offsets in one program and use the same offsets in another program. Program *G92* in the first program. This will set parameters 5211 to 5219. Do not use *G92.1* in the remainder of the first program. The parameter values will be saved when the first program exits and restored when the second one starts up. Use *G92.3* near the beginning of the second program. That will restore the offsets saved in the first program.

6.3.59 G93, G94, G95 Feed Rate Mode

- *G93* - is Inverse Time Mode. In inverse time feed rate mode, an F word means the move should be completed in [one divided by the F number] minutes. For example, if the F number is 2.0, the move should be completed in half a minute.

When the inverse time feed rate mode is active, an F word must appear on every line which has a G1, G2, or G3 motion, and an F word on a line that does not have G1, G2, or G3 is ignored. Being in inverse time feed rate mode does not affect G0 ([rapid move](#)) motions.

- *G94* - is Units per Minute Mode. In units per minute feed mode, an F word is interpreted to mean the controlled point should move at a certain number of inches per minute, millimeters per minute, or degrees per minute, depending upon what length units are being used and which axis or axes are moving.
- *G95* - is Units per Revolution Mode In units per revolution mode, an F word is interpreted to mean the controlled point should move a certain number of inches per revolution of the spindle, depending on what length units are being used and which axis or axes are moving. *G95* is not suitable for threading, for threading use *G33* or *G76*. *G95* requires that spindle.N.speed-in to be connected. The actual spindle to which the feed is synchronised is chosen by the \$ parameter

It is an error if:

- Inverse time feed mode is active and a line with G1, G2, or G3 (explicitly or implicitly) does not have an F word.
 - A new feed rate is not specified after switching to *G94* or *G95*
-

6.3.60 G96, G97 Spindle Control Mode

G96 <D-> S- <\$-> (Constant Surface Speed Mode)
 G97 S- <\$-> (RPM Mode)

1. *D* - maximum rotation speed (RPM), optional
2. *S* - spindle speed
3. *\$* - the spindle of which the speed will be varied, optional.
 - *G96 S- <D->* - selects constant surface speed of *S*:
 - In feet per minute if G20 is in effect,
 - or meters per minute if G21 is in effect.

When using G96, ensure that X0 in the current coordinate system (including offsets and tool lengths) is the center of rotation or LinuxCNC will not give the desired surface speed. G96 is not affected by radius or diameter mode.

To achieve CSS mode on selected spindles programme successive G96 commands for each spindle prior to issuing M3.

- *G97* selects RPM mode.

G96 Example Line

G96 D2500 S250 (set CSS with a max rpm of 2500 and a surface speed of 250)

It is an error if:

- *S* is not specified with G96
- A feed move is specified in G96 mode while the spindle is not turning

6.3.61 G98, G99 Canned Cycle Return Level

When spindle retracts during canned cycles, there are two options to choose from for the way it does it:

- *G98* - retract to the position that axis was in just before this series of one or more contiguous canned cycles was started.
- *G99* - retract to the position specified by the R word of the canned cycle.

Program a *G98* and the canned cycle will use the Z position prior to the canned cycle as the Z return position if it is higher than the R value specified in the cycle. If it is lower, the R value will be used. The R word has different meanings in absolute distance mode and incremental distance mode.

G98 Retract to Origin

G0 X1 Y2 Z3
 G90 G98 G81 X4 Y5 Z-0.6 R1.8 F10

The G98 to the second line above means that the return move will be to the value of Z in the first line since it is higher than the R value specified.

The *initial* (G98) plane is reset any time cycle motion mode is abandoned, whether explicitly (G80) or implicitly (any motion code that is not a cycle). Switching among cycle modes (say G81 to G83) does NOT reset the *initial* plane. It is possible to switch between G98 and G99 during a series of cycles.

6.4 M-Codes

6.4.1 M-Code Quick Reference Table

Code	Description
M0 M1	Program Pause
M2 M30	Program End
M60	Pallet Change Pause
M3 M4 M5	Spindle Control
M6	Tool Change
M7 M8 M9	Coolant Control
M19	Orient Spindle
M48 M49	Feed & Spindle Overrides Enable/Disable
M50	Feed Override Control
M51	Spindle Override Control
M52	Adaptive Feed Control
M53	Feed Stop Control
M61	Set Current Tool Number
m62-m65	Output Control
M66	Input Control
M67	Analog Output Control
M68	Analog Output Control
M70	Save Modal State
M71	Invalidate Stored Modal State
M72	Restore Modal State
M73	Save Autorestore Modal State
M98 M99	Call and Return From Subprogram
M100-M199	User Defined M-Codes

6.4.2 M0, M1 Program Pause

- *M0* - pause a running program temporarily. LinuxCNC remains in the Auto Mode so MDI and other manual actions are not enabled. Pressing the resume button will restart the program at the following line.
- *M1* - pause a running program temporarily if the optional stop switch is on. LinuxCNC remains in the Auto Mode so MDI and other manual actions are not enabled. Pressing the resume button will restart the program at the following line.

Note

It is OK to program *M0* and *M1* in MDI mode, but the effect will probably not be noticeable, because normal behavior in MDI mode is to stop after each line of input anyway.

6.4.3 M2, M30 Program End

- *M2* - end the program. Pressing Cycle Start ("R" in the Axis GUI) will restart the program at the beginning of the file.
 - *M30* - exchange pallet shuttles and end the program. Pressing Cycle Start will start the program at the beginning of the file.
-

Both of these commands have the following effects:

- Change from Auto mode to MDI mode.
- Origin offsets are set to the default (like *G54*).
- Selected plane is set to XY plane (like *G17*).
- Distance mode is set to absolute mode (like *G90*).
- Feed rate mode is set to units per minute (like *G94*).
- Feed and speed overrides are set to ON (like *M48*).
- Cutter compensation is turned off (like *G40*).
- The spindle is stopped (like *M5*).
- The current motion mode is set to feed (like *G1*).
- Coolant is turned off (like *M9*).

Note

Lines of code after M2/M30 will not be executed. Pressing Cycle Start will start the program at the beginning of the file.

**Warning**

Using % to wrap the G-code does not do the same thing as a *Program End*. See the section on [File Requirements](#) for more information on what using % does not do.

6.4.4 M60 Pallet Change Pause

- *M60* - exchange pallet shuttles and then pause a running program temporarily (regardless of the setting of the optional stop switch). Pressing the cycle start button will restart the program at the following line.

6.4.5 M3, M4, M5 Spindle Control

- *M3* [*\$n*] - start the selected spindle clockwise at the *S* speed.
- *M4* [*\$n*] - start the selected spindle counterclockwise at the *S* speed.
- *M5* [*\$n*] - stop the selected spindle.

Use \$ to operate on specific spindles. If \$ is omitted then the commands default to operating on spindle 0. Use \$-1 to operate on all active spindles.

This example will start spindles 0, 1, and 2 simultaneously at different speeds:

```
S100 $0
S200 $1
S300 $2
M3 $-1
```

This example will then reverse spindle 1 but leave the other spindles rotating forwards:

M4 \$1

And this will stop spindle 2 and leave the other spindles rotating:

M5 \$2

If the \$ is omitted then behaviour is exactly as normal for a single spindle machine.

It is OK to use *M3* or *M4* if the [S](#) spindle speed is set to zero. If this is done (or if the speed override switch is enabled and set to zero), the spindle will not start turning. If, later, the spindle speed is set above zero (or the override switch is turned up), the spindle will start turning. It is OK to use *M3* or *M4* when the spindle is already turning or to use *M5* when the spindle is already stopped.

6.4.6 M6 Tool Change

6.4.6.1 Manual Tool Change

If the HAL component *hal_manualtoolchange* is loaded, M6 will stop the spindle and prompt the user to change the tool based on the last *T*-number programmed. For more information on *hal_manualtoolchange* see the [Manual Tool Change](#) section.

6.4.6.2 Tool Changer

To change a tool in the spindle from the tool currently in the spindle to the tool most recently selected (using a *T* word - see section [Select Tool](#)), program *M6*. When the tool change is complete:

- The spindle will be stopped.
- The tool that was selected (by a *T* word on the same line or on any line after the previous tool change) will be in the spindle.
- If the selected tool was not in the spindle before the tool change, the tool that was in the spindle (if there was one) will be placed back into the tool changer magazine.
- If configured in the INI file some axis positions may move when a *M6* is issued. See the [EMCIO section](#) for more information on tool change options.
- No other changes will be made. For example, coolant will continue to flow during the tool change unless it has been turned off by an *M9*.

Note

The *T*-word is an integer number designating the tool pocket number in the carousel (not its index).



Warning

The tool length offset is not changed by *M6*, use [G43](#) after the *M6* to change the tool length offset.

The tool change may include axis motion. It is OK (but not useful) to program a change to the tool already in the spindle. It is OK if there is no tool in the selected slot; in that case, the spindle will be empty after the tool change. If slot zero was last selected, there will definitely be no tool in the spindle after a tool change. The tool changer will have to be setup to perform the tool change in HAL and possibly ClassicLadder.

6.4.7 M7, M8, M9 Coolant Control

- *M7* - turn mist coolant on. M7 controls iocontrol.0.coolant-mist pin.
- *M8* - turn flood coolant on. M8 controls iocontrol.0.coolant-flood pin.
- *M9* - turn both M7 and M8 off.

Connect one or both of the coolant control pins in HAL before M7 or M8 will control an output. M7 and M8 can be used to turn on any output via G-code.

It is OK to use any of these commands, regardless of the current coolant state.

6.4.8 M19 Orient Spindle

M19 R- Q- [P-] [\$-]

- *R* Position to rotate to from 0, valid range is 0-360 degrees
- *Q* Number of seconds to wait until orient completes. If spindle.N.is-oriented does not become true within Q timeout an error occurs.
- *P* Direction to rotate to position.
 - 0 rotate for smallest angular movement (default)
 - 1 always rotate clockwise (same as M3 direction)
 - 2 always rotate counterclockwise (same as M4 direction)
- \$ The spindle to orient (actually only determines which HAL pins carry the spindle position commands)

M19 is a command of modal group 7, like M3, M4 and M5. M19 is cleared by any of M3,M4,M5.

Spindle orientation requires a quadrature encoder with an index to sense the spindle shaft position and direction of rotation.

INI Settings in the [RS274NGC] section:

- ORIENT_OFFSET = 0-360 (fixed offset in degrees added to M19 R word)
- HAL Pins
 - *spindle.N.orient-angle* (out float) Desired spindle orientation for M19. Value of the M19 R word parameter plus the value of the [RS274NGC]ORIENT_OFFSET INI parameter.
 - *spindle.N.orient-mode* (out s32) Desired spindle rotation mode. Reflects M19 P parameter word, default = 0.
 - *spindle.N.orient* (out bit) Indicates start of spindle orient cycle. Set by M19. Cleared by any of M3,M4,M5. If spindle-orient-fault is not zero during spindle-orient true, the M19 command fails with an error message.
 - *spindle.N.is-oriented* (in bit) Acknowledge pin for spindle-orient. Completes orient cycle. If spindle-orient was true when spindle-is-oriented was asserted, the spindle-orient pin is cleared and the spindle-locked pin is asserted. Also, the spindle-brake pin is asserted.
 - *spindle.N.orient-fault* (in s32) Fault code input for orient cycle. Any value other than zero will cause the orient cycle to abort.
 - *spindle.N.locked* (out bit) Spindle orient complete pin. Cleared by any of M3,M4,M5.

6.4.9 M48, M49 Speed and Feed Override Control

- *M48* - enable the spindle speed and feed rate override controls.
- *M49* - disable both controls.

These commands also take an optional *\$* parameter to determine which spindle they operate on.

It is OK to enable or disable the controls when they are already enabled or disabled. See the [Feed Rate](#) section for more details.

They also can be be toggled individually using *M50* and *M51*, see below.

6.4.10 M50 Feed Override Control

- *M50 <P1>* - enable the feed rate override control. The P1 is optional.
- *M50 P0* - disable the feed rate control.

While disabled the feed override will have no influence, and the motion will be executed at programmed feed rate. (unless there is an adaptive feed rate override active).

6.4.11 M51 Spindle Speed Override Control

- *M51 <P1> <\$->* - enable the spindle speed override control for the selected spindle. The P1 is optional.
- *M51 P0 <\$->* - disable the spindle speed override control program.

While disabled the spindle speed override will have no influence, and the spindle speed will have the exact program specified value of the S-word (described in the [Spindle Speed](#) section).

6.4.12 M52 Adaptive Feed Control

- *M52 <P1>* - use an adaptive feed. The P1 is optional.
- *M52 P0* - stop using adaptive feed.

When adaptive feed is enabled, some external input value is used together with the user interface feed override value and the commanded feed rate to set the actual feed rate. In LinuxCNC, the HAL pin *motion.adaptive-feed* is used for this purpose. Values on *motion.adaptive-feed* should range from -1 (programmed speed in reverse) to 1 (full speed). 0 is equivalent to feed-hold.

Note

The use of negative adaptive-feed for reverse run is a new feature and is not very well tested as yet. The intended use is for plasma cutters and wire spark eroders but it is not limited to such applications.

6.4.13 M53 Feed Stop Control

- *M53 <P1>* - enable the feed stop switch. The P1 is optional. Enabling the feed stop switch will allow motion to be interrupted by means of the feed stop control. In LinuxCNC, the HAL pin *motion.feed-hold* is used for this purpose. A *true* value will cause the motion to stop when *M53* is active.
 - *M53 P0* - disable the feed stop switch. The state of *motion.feed-hold* will have no effect on feed when *M53* is not active.
-

6.4.14 M61 Set Current Tool

- *M61 Q-* - change the current tool number while in MDI or Manual mode without a tool change. One use is when you power up LinuxCNC with a tool currently in the spindle you can set that tool number without doing a tool change.



Warning

The tool length offset is not changed by *M61*, use *G43* after the *M61* to change the tool length offset.

It is an error if:

- *Q-* is not 0 or greater

6.4.15 M62 - M65 Digital Output Control

- *M62 P-* - turn on digital output synchronized with motion.
- *M63 P-* - turn off digital output synchronized with motion.
- *M64 P-* - turn on digital output immediately.
- *M65 P-* - turn off digital output immediately.

The *P-* word specifies the digital output number. The *P-*word ranges from 0 to a default value of 3. If needed the the number of I/O can be increased by using the `num_dio` parameter when loading the motion controller. See the [Motion](#) section for more information.

The *M62* & *M63* commands will be queued. Subsequent commands referring to the same output number will overwrite the older settings. More than one output change can be specified by issuing more than one *M62/M63* command.

The actual change of the specified outputs will happen at the beginning of the next motion command. If there is no subsequent motion command, the queued output changes won't happen. It's best to always program a motion G-code (*G0*, *G1*, etc) right after the *M62/63*.

M64 & *M65* happen immediately as they are received by the motion controller. They are not synchronized with movement, and they will break blending.

Note

M62-65 will not function unless the appropriate `motion.digital-out-nn` pins are connected in your HAL file to outputs.

6.4.16 M66 Wait on Input

M66 P- | E- <L->

- *P-* - specifies the digital input number from 0 to 3.
- *E-* - specifies the analog input number from 0 to 3.
- *L-* - specifies the wait mode type.

- *Mode 0: IMMEDIATE* - no waiting, returns immediately. The current value of the input is stored in parameter #5399
- *Mode 1: RISE* - waits for the selected input to perform a rise event.
- *Mode 2: FALL* - waits for the selected input to perform a fall event.
- *Mode 3: HIGH* - waits for the selected input to go to the HIGH state.
- *Mode 4: LOW* - waits for the selected input to go to the LOW state.
- Q- - specifies the timeout in seconds for waiting. If the timeout is exceeded, the wait is interrupt, and the variable #5399 will be holding the value -1. The Q value is ignored if the L-word is zero (IMMEDIATE). A Q value of zero is an error if the L-word is non-zero.
- Mode 0 is the only one permitted for an analog input.

M66 Example Lines

```
M66 P0 L3 Q5 (wait up to 5 seconds for digital input 0 to turn on)
```

M66 wait on an input stops further execution of the program, until the selected event (or the programmed timeout) occurs.

It is an error to program M66 with both a P-word and an E-word (thus selecting both an analog and a digital input). In LinuxCNC these inputs are not monitored in real time and thus should not be used for timing-critical applications.

The number of I/O can be increased by using the num_dio or num_aio parameter when loading the motion controller. See the [Motion](#) section for more information.

Note

M66 will not function unless the appropriate motion.digital-in-nn pins or motion.analog-in-nn pins are connected in your HAL file to an input.

Example HAL Connection

```
net signal-name motion.digital-in-00 <= parport.0.pin10-in
```

6.4.17 M67 Analog Output,Synchronized

```
M67 E- Q-
```

- *M67* - set an analog output synchronized with motion.
- *E-* - output number ranging from 0 to 3.
- *Q-* - is the value to set (set to 0 to turn off).

The actual change of the specified outputs will happen at the beginning of the next motion command. If there is no subsequent motion command, the queued output changes won't happen. It's best to always program a motion G-code (G0, G1, etc) right after the M67. M67 functions the same as M62-63.

The number of I/O can be increased by using the num_dio or num_aio parameter when loading the motion controller. See the [Motion](#) section for more information.

Note

M67 will not function unless the appropriate motion.analog-out-nn pins are connected in your HAL file to outputs.

6.4.18 M68 Analog Output, Immediate

M68 E- Q-

- *M68* - set an analog output immediately.
- *E-* - output number ranging from 0 to 3.
- *Q-* - is the value to set (set to 0 to turn off).

M68 output happen immediately as they are received by the motion controller. They are not synchronized with movement, and they will break blending. M68 functions the same as M64-65.

The number of I/O can be increased by using the `num_dio` or `num_aio` parameter when loading the motion controller. See the [Motion](#) section for more information.

Note

M68 will not function unless the appropriate `motion.analog-out-nn` pins are connected in your HAL file to outputs.

6.4.19 M70 Save Modal State

To explicitly save the modal state at the current call level, program *M70*. Once modal state has been saved with *M70*, it can be restored to exactly that state by executing an *M72*.

A pair of *M70* and *M72* instructions will typically be used to protect a program against inadvertent modal changes within subroutines.

M70 Saved stateM70 Saved state

The state saved consists of:

- current G20/G21 settings (imperial/metric)
 - selected plane (G17/G18/G19 G17.1,G18.1,G19.1)
 - status of cutter compensation (G40,G41,G42,G41.1,G42,1)
 - distance mode - relative/absolute (G90/G91)
 - feed mode (G93/G94,G95)
 - current coordinate system (G54-G59.3)
 - tool length compensation status (G43,G43.1,G49)
 - retract mode (G98,G99)
 - spindle mode (G96-css or G97-RPM)
 - arc distance mode (G90.1, G91.1)
 - lathe radius/diameter mode (G7,G8)
 - path control mode (G61, G61.1, G64)
 - current feed and speed (*F* and *S* values)
 - spindle status (M3,M4,M5) - on/off and direction
-

- mist (M7) and flood (M8) status
- speed override (M51) and feed override (M50) settings
- adaptive feed setting (M52)
- feed hold setting (M53)

Note that in particular, the motion mode (G1 etc) is NOT restored.

current call level means either:

- executing in the main program. There is a single storage location for state at the main program level; if several *M70* instructions are executed in turn, only the most recently saved state is restored when an *M72* is executed.
- executing within a G-code subroutine. The state saved with *M70* within a subroutine behaves exactly like a local named parameter - it can be referred to only within this subroutine invocation with an *M72* and when the subroutine exits, the parameter goes away.

A recursive invocation of a subroutine introduces a new call level.

6.4.20 M71 Invalidate Stored Modal State

Modal state saved with an *M70* or by an *M73* at the current call level is invalidated (cannot be restored from anymore).

A subsequent *M72* at the same call level will fail.

If executed in a subroutine which protects modal state by an *M73*, a subsequent return or endsub will **not** restore modal state.

The usefulness of this feature is dubious. It should not be relied upon as it might go away.

6.4.21 M72 Restore Modal State

Modal state saved with an *M70* code can be restored by executing an *M72*.

The handling of G20/G21 is specially treated as feeds are interpreted differently depending on G20/G21: if length units (mm/in) are about to be changed by the restore operation, 'M72' will restore the distance mode first, and then all other state including feed to make sure the feed value is interpreted in the correct unit setting.

It is an error to execute an *M72* with no previous *M70* save operation at that level.

The following example demonstrates saving and explicitly restoring modal state around a subroutine call using *M70* and *M72*. Note that the *imperialsub* subroutine is not "aware" of the M7x features and can be used unmodified:

```
0<showstate> sub
(DEBUG, imperial=#<_imperial> absolute=#<_absolute> feed=#<_feed> rpm=#<_rpm>)
0<showstate> endsub

0<imperialsub> sub
g20 (imperial)
g91 (relative mode)
F5 (low feed)
S300 (low rpm)
(debug, in subroutine, state now:)
0<showstate> call
```

```

0<imperialsub> endsub

; main program
g21 (metric)
g90 (absolute)
f200 (fast speed)
S2500 (high rpm)

(debug, in main, state now:)
o<showstate> call

M70 (save caller state in at global level)
0<imperialsub> call
M72 (explicitly restore state)

(debug, back in main, state now:)
o<showstate> call
m2

```

6.4.22 M73 Save and Autorestore Modal State

To save modal state within a subroutine, and restore state on subroutine *endsub* or any *return* path, program *M73*.

Aborting a running program in a subroutine which has an *M73* operation will **not** restore state .

Also, the normal end (*M2*) of a main program which contains an *M73* will **not** restore state.

The suggested use is at the beginning of a O-word subroutine as in the following example. Using *M73* this way enables designing subroutines which need to modify modal state but will protect the calling program against inadvertent modal changes. Note the use of [predefined named parameters](#) in the *showstate* subroutine.

```

0<showstate> sub
(DEBUG, imperial=#<_imperial> absolute=#<_absolute> feed=#<_feed> rpm=#<_rpm>)
0<showstate> endsub

0<imperialsub> sub
M73 (save caller state in current call context, restore on return or endsub)
g20 (imperial)
g91 (relative mode)
F5 (low feed)
S300 (low rpm)
(debug, in subroutine, state now:)
o<showstate> call

; note - no M72 is needed here - the following endsub or an
; explicit 'return' will restore caller state
0<imperialsub> endsub

; main program
g21 (metric)
g90 (absolute)
f200 (fast speed)
S2500 (high rpm)
(debug, in main, state now:)
o<showstate> call
0<imperialsub> call
(debug, back in main, state now:)
o<showstate> call
m2

```

6.4.23 M98 and M99

The interpreter supports Fanuc-style main- and sub-programs with the *M98* and *M99* M-codes. See [Fanuc-Style Programs](#).

6.4.23.1 Selectively Restoring Modal State

Executing an *M72* or returning from a subroutine which contains an *M73* will restore [all modal state saved](#).

If only some aspects of modal state should be preserved, an alternative is the usage of [predefined named parameters](#), local parameters and conditional statements. The idea is to remember the modes to be restored at the beginning of the subroutine, and restore these before exiting. Here is an example, based on snippet of *nc_files/tool-length-probe.ngc*:

```
0<measure> sub    (measure reference tool)
;
#<absolute> = #<_absolute>  (remember in local variable if G90 was set)
;
g30 (above switch)
g38.2 z0 f15 (measure)
g91 g0z.2 (off the switch)
#1000=#5063 (save reference tool length)
(print,reference length is #1000)
;
0<restore_abs> if [#<absolute>]
    g90 (restore G90 only if it was set on entry:)
0<restore_abs> endif
;
0<measure> endsub
```

6.4.24 M100-M199 User Defined Commands

M1-- <P- Q->

- *M1--* - an integer in the range of 100 - 199.
- *P-* - a number passed to the file as the first parameter.
- *Q-* - a number passed to the file as the second parameter.

Note

After creating a new *M1nn* file you must restart the GUI so it is aware of the new file, otherwise you will get an *Unknown m code* error.

The external program named *M100* through *M199* (no extension, a capital M, found in directory pointed by *[DISPLAY] PROGRAM_PREFIX* parameter of the INI file) is executed with the optional P and Q values as its two arguments.

Execution of the G-code file pauses until the external program exits. Any valid executable file can be used. The file must be located in the search path specified in the INI file configuration. See the [Display](#) section for more information on search paths.

After creating a new *M1nn* program, the GUI should be restarted so that the new program is taken into account, otherwise a *Unknown M-code* error will occur.

**Warning**

Do not use a word processor to create or edit the files. A word processor will leave unseen codes that will cause problems and may prevent a bash or python file from working. Use a text editor like Geany in Debian or Notepad++ in other operating systems to create or edit the files.

The error *Unknown M-code used* denotes one of the following:

- The specified User Defined Command does not exist
- The file is not an executable file
- The file name has an extension
- The file name does not follow this format Mnnn where nnn = 100 through 199
- The file name used a lower case M

For example to open and close a collet closer that is controlled by a parallel port pin using a bash script file using M101 and M102. Create two files named M101 and M102. Set them as executable files (typically right click/properties/permissions) before running LinuxCNC. Make sure the parallel port pin is not connected to anything in a HAL file.

M101 Example File

```
#!/bin/bash
# file to turn on parport pin 14 to open the collet closer
halcmd setp parport.0.pin-14-out True
exit 0
```

M102 Example File

```
#!/bin/bash
# file to turn off parport pin 14 to open the collet closer
halcmd setp parport.0.pin-14-out False
exit 0
```

To pass a variable to a M1nn file you use the P and Q option like this:

```
M100 P123.456 Q321.654
```

M100 Example file

```
#!/bin/bash
voltage=$1
feedrate=$2
halcmd setp thc.voltage $voltage
halcmd setp thc.feedrate $feedrate
exit 0
```

To display a graphic message and stop until the message window is closed use a graphic display program like Eye of Gnome to display the graphic file. When you close it the program will resume.

M110 Example file

```
#!/bin/bash
eog /home/john/linuxcnc/nc_files/message.png
exit 0
```

To display a graphic message and continue processing the G-code file suffix an ampersand to the command.

M110 Example display and keep going

```
#!/bin/bash
eog /home/john/linuxcnc/nc_files/message.png &
exit 0
```

6.5 O Codes

6.5.1 Use of O-codes

O-codes provide for flow control in NC programs. Each block has an associated number, which is the number used after O. Care must be taken to properly match the O-numbers. O codes use the letter *O* not the number zero as the first character in the number like O100 or o100.

6.5.2 Numbering

Numbered O codes must have a unique number for each subroutine,

Numbering Example

```
(the start of o100)
o100 sub
(notice that the if-endif block uses a different number)
  (the start of o110)
  o110 if [#2 GT 5]
    (some code here)
  (the end of o110)
  o110 endif
  (some more code here)
(the end of o100)
o100 endsub
```

6.5.3 Comments

Comments on the same line as the O word should not be used as the behavior can change in the future. The behavior is undefined if:

- The same number is used for more than one block.
- Other words are used on a line with an O- word.
- Comments are used on a line with an O-word.

Note

Using the lower case o makes it easier to distinguish from a 0 that might have been mistyped. For example *o100* is easier to see than *O100* that it is not a 0.

6.5.4 Subroutines

Subroutines starts at *Onnn sub* and ends at *Onnn endsub*. The lines between *Onnn sub* and *Onnn endsub* are not executed until the subroutine is called with *Onnn call*. Each subroutine must use a unique number.

Subroutine Example

```
o100 sub
  G53 G0 X0 Y0 Z0 (rapid move to machine home)
o100 endsub

(the subroutine is called)
o100 call
M2
```

See [G53](#), [G0](#) and [M2](#) sections for more information.

O- Return Inside a subroutine, *O- return* can be executed. This immediately returns to the calling code, just as though *O- endsub* was encountered.

O- Return Example

```
o100 sub
  (test if parameter #2 is greater than 5)
  o110 if [#2 GT 5]
    (return to top of subroutine if test is true)
    o100 return
  o110 endif
  (this only gets executed if parameter #2 is not greater than 5)
  (DEBUG, parameter 1 is [#1])
o100 endsub
```

See the [Binary Operators](#) and [Parameters](#) sections for more information.

O- Call *O- Call* takes up to 30 optional arguments, which are passed to the subroutine as *#1*, *#2*, ..., *#N*. Parameters from *#N+1* to *#30* have the same value as in the calling context. On return from the subroutine, the values of parameters *#1* through *#30* (regardless of the number of arguments) will be restored to the values they had before the call. Parameters *#1* - *#30* are local to the subroutine.

Because *1 2 3* is parsed as the number 123, the parameters must be enclosed in square brackets. The following calls a subroutine with 3 arguments:

O- Call Example

```
o100 sub
  (test if parameter #2 is greater than 5)
  o110 if [#2 GT 5]
    (return to top of subroutine if test is true)
    o100 return
  o110 endif
  (this only gets executed if parameter #2 is not greater than 5)
  (DEBUG, parameter 1 is [#1])
  (DEBUG, parameter 3 is [#3])
o100 endsub

o100 call [100] [2] [325]
```

Subroutine bodies may not be nested. They may only be called after they are defined. They may be called from other functions, and may call themselves recursively if it makes sense to do so. The maximum subroutine nesting level is 10.

Subroutines can change the value of parameters above *#30* and those changes will be visible to the calling code. Subroutines may also change the value of global named parameters.

6.5.4.1 Fanuc-Style Numbered Programs

Numbered programs (both main and subprograms), the *M98* call and *M99* return M-codes, and their respective semantic differences are an alternative to the rs274ngc subroutines described above, provided for compatibility with Fanuc and other machine controllers.

Numbered programs are enabled by default, and may be disabled by placing `DISABLE_FANUC_STYLE_SUB = 1` in the `[RS274NGC]` section of the `.ini` file.

Note

Numbered main and subprogram definitions and calls differ from traditional rs274ngc both in syntax and execution. To reduce the possibility of confusion, the interpreter will raise an error if definitions of one style are mixed with calls of another.

Numbered Subprogram Simple Example

```
o1 (Example 1)      ; Main program 1, "Example 1"
M98 P100            ; Call subprogram 100
M30                 ; End main program

o100                ; Beginning of subprogram 100
  G53 G0 X0 Y0 Z0   ; Rapid move to machine home
M99                 ; Return from subprogram 100
```

o1 (Title) The optional main program beginning block gives the main program the number 1. Some controllers treat an optional following parenthesized comment as a program title, `Example 1` in this example, but this has no special meaning in the rs274ngc interpreter.

M98 P- <L-> Call a numbered subprogram. The block `M98 P100` is analogous to the traditional `o100 call` syntax, but may only be used to call a following numbered subprogram defined with `o100...M99`. An optional *L*-word specifies a loop count.

M30 The main program must be terminated with `M02` or `M30` (or `M99`; see below).

O- subprogram definition start Marks the start of a numbered subprogram definition. The block `O100` is similar to `o100 sub`, except that it must be placed later in the file than the `M98 P100` calling block.

M99 return from numbered subroutine The block `M99` is analogous to the traditional `o100 endsub` syntax, but may only terminate a numbered program (`o100` in this example), and may not terminate a subroutine beginning with the `o100 sub` syntax.

The `M98` subprogram call differs from rs274ngc `o call` in the following ways:

- The numbered subprogram must follow the `M98` call in the program file. The interpreter will throw an error if the subprogram precedes the call block.
- Parameters `#1`, `#2`, ..., `#30` are global and accessible in numbered subprograms, similar to higher-numbered parameters in traditional style calls. Modifications to these parameters within a subprogram are global modifications, and will be persist after subprogram return.
- `M98` subprogram calls have no return value.
- `M98` subprogram call blocks may contain an optional *L*-word specifying a loop repeat count. Without the *L*-word, the subprogram will execute once only (equivalent to `M98 L1`). An `M98 L0` block will not execute the subprogram.

In rare cases, the `M99` M-code may be used to terminate the main program, where it indicates an *endless program*. When the interpreter reaches an `M99` in the main program, it will skip back to the beginning of the file and resume execution at the first line. An example use of an endless program is

in a machine warm-up cycle; a block delete program end /M30 block might be used to stop the cycle at a tidy point when the operator is ready.

Numbered Subprogram Full Example

```

01                                ; Main program 1
  #1 = 0
  (PRINT,X MAIN BEGIN: 1=#1)
  M98 P100 L5                    ; Call subprogram 100
  (PRINT,X MAIN END: 1=#1)
M30                                ; End main program

0100                             ; Subprogram 100
  #1 = [#1 + 1]
  M98 P200 L5                    ; Call subprogram 200
  (PRINT,>> 0100: #1)
M99                                ; Return from Subprogram 100

0200                             ; Subprogram 200
  #1 = [#1 + 0.01]
  (PRINT,>>>> 0200: #1)
M99                                ; Return from Subprogram 200

```

In this example, parameter #1 is initialized to 0. Subprogram 0100 is called five times in a loop. Nested within each call to 0100, subprogram 0200 is called five times in a loop, for 25 times total.

Note that parameter #1 is global. At the end of the main program, after updates within 0100 and 0200, its value will equal 5.25.

6.5.5 Looping

The *while* loop has two structures: *while/endwhile*, and *do/while*. In each case, the loop is exited when the *while* condition evaluates to false. The difference is when the test condition is done. The *do/while* loop runs the code in the loop then checks the test condition. The *while/endwhile* loop does the test first.

While Endwhile Example

```

(draw a sawtooth shape)
G0 X1 Y0 (move to start position)
#1 = 0 (assign parameter #1 the value of 0)
F25 (set a feed rate)
o101 while [#1 LT 10]
  G1 X0
  G1 Y[#1/10] X1
  #1 = [#1+1] (increment the test counter)
o101 endwhile
M2 (end program)

```

Do While Example

```

#1 = 0 (assign parameter #1 the value of 0)
o100 do
  (debug, parameter 1 = #1)
  o110 if [#1 EQ 2]
    #1 = 3 (assign the value of 3 to parameter #1)
    (msg, #1 has been assigned the value of 3)
    o100 continue (skip to start of loop)
  o110 endif
  (some code here)
  #1 = [#1 + 1] (increment the test counter)

```

```
o100 while [#1 LT 3]
(msg, Loop Done!)
M2
```

Inside a while loop, *O- break* immediately exits the loop, and *O- continue* immediately skips to the next evaluation of the *while* condition. If it is still true, the loop begins again at the top. If it is false, it exits the loop.

6.5.6 Conditional

The *if* conditional consists of a group of statements with the same *o* number that start with *if* and end with *endif*. Optional *elseif* and *else* conditions may be between the starting *if* and the ending *endif*.

If the *if* conditional evaluates to true then the group of statements following the *if* up to the next conditional line are executed.

If the *if* conditional evaluates to false then the *elseif* conditions are evaluated in order until one evaluates to true. If the *elseif* condition is true then the statements following the *elseif* up to the next conditional line are executed. If none of the *if* or *elseif* conditions evaluate to true then the statements following the *else* are executed. When a condition is evaluated to true no more conditions are evaluated in the group.

If Endif Example

```
(if parameter #31 is equal to 3 set S2000)
o101 if [#31 EQ 3]
    S2000
o101 endif
```

If Elseif Else Endif Example

```
(if parameter #2 is greater than 5 set F100)
o102 if [#2 GT 5]
    F100
o102 elseif [#2 LT 2]
    (else if parameter #2 is less than 2 set F200)
    F200
    (else if parameter #2 is 2 through 5 set F150)
o102 else
    F150
o102 endif
```

Several conditions may be tested for by *elseif* statements until the *else* path is finally executed if all preceding conditions are false:

If Elseif Else Endif Example

```
(if parameter #2 is greater than 5 set F100)
O102 if [#2 GT 5]
    F100
    (else if parameter #2 less than 2 set F200)
O102 elseif [#2 LT 2]
    F20
    (parameter #2 is between 2 and 5)
O102 else
    F200
O102 endif
```

6.5.7 Repeat

The *repeat* will execute the statements inside of the repeat/endrepeat the specified number of times. The example shows how you might mill a diagonal series of shapes starting at the present position.

Example with repeat

```
(Mill 5 diagonal shapes)
G91 (Incremental mode)
o103 repeat [5]
... (insert milling code here)
G0 X1 Y1 (diagonal move to next position)
o103 endrepeat
G90 (Absolute mode)
```

6.5.8 Indirection

The O-number may be given by a parameter and/or calculation.

Indirection Example

```
o[#101+2] call
```

Computing values in O-words For more information on computing values see the following sections:

- [Parameters](#)
- [Expressions](#)
- [Binary Operators](#)
- [Functions](#)

6.5.9 Calling Files

To call a separate file with a subroutine name the file the same as your call and include a sub and endsub in the file. The file must be in the directory pointed to by *PROGRAM_PREFIX* or *SUBROUTINE_PATH* in the INI file. The file name can include **lowercase** letters, numbers, dash, and underscore only. A named subroutine file can contain only a single subroutine definition.

Named File Example

```
o<myfile> call
```

Numbered File Example

```
o123 call
```

In the called file you must include the oxxx sub and endsub and the file must be a valid file.

Called File Example

```
(filename myfile.ngc)
o<myfile> sub
  (code here)
o<myfile> endsub
M2
```

Note

The file names are lowercase letters only so *o<MyFile>* is converted to *o<myfile>* by the interpreter. More information about the search path and options for the search path are in the INI configuration section.

6.5.10 Subroutine return values

Subroutines may optionally return a value by an optional expression at an *endsub* or *return* statement.

Return value example

```
o123 return [#2 *5]
...
o123 endsub [3 * 4]
```

A subroutine return value is stored in the `<_value>` [predefined named parameter](#), and the `<_value_returned` predefined parameter is set to 1, to indicate a value was returned. Both parameters are global, and are cleared just before the next subroutine call.

6.5.11 Errors

The following statements cause an error message and abort the interpreter:

- a return or endsub not within a sub definition
- a label on repeat which is defined elsewhere
- a label on while which is defined elsewhere and not referring to a do
- a label on if defined elsewhere
- a undefined label on else or elseif
- a label on else, elseif or endif not pointing to a matching if
- a label on break or continue which does not point to a matching while or do
- a label on endrepeat or endwhile no referring to a corresponding while or repeat

To make these errors non-fatal warnings on stderr, set bit 0x20 in the `[RS274NGC]FEATURE= mask` ini option.

6.6 Other Codes

6.6.1 F: Set Feed Rate

Fx - set the feed rate to *x*. *x* is usually in machine units (inches or millimeters) per minute.

The application of the feed rate is as described in the [Feed Rate](#) Section, unless *inverse time feed rate mode* or *feed per revolution mode* are in effect, in which case the feed rate is as described in the [G93 G94 G95](#) section.

6.6.2 S: Set Spindle Speed

Sx [\$n] - set the speed of the spindle to *x* revolutions per minute (RPM) with the optional *\$* set the spindle speed for a specific spindle. Without the *\$* the command will default to spindle.0.

The spindle(s) or selected spindle will turn at that speed when a *M3* or *M4* is in effect. It is OK to program an *S* word whether the spindle is turning or not. If the speed override switch is enabled and not set at 100%, the speed will be different from what is programmed.

It is OK to program *S0*, the spindle will not turn if that is done.

It is an error if:

- the S number is negative.

As described in the section [Right-hand Tapping Cycle with Dwell](#), if a *G84* (tapping) drilling cycle is active and the speed and feed potentiometers are enabled, the one with the lowest setting will be used. The rotational speed and feed rate will remain synchronized. In this case, the speed may differ from the one programmed, even if the speed correction potentiometer is set to 100%.

6.6.3 T: Select Tool

*T*x - prepare to change to tool x.

The tool is not changed until an *M6* is programmed (see Section [M6](#)). The *T* word may appear on the same line as the *M6* or on a previous line. It is OK if *T* words appear on two or more lines with no tool change. Only the the most recent *T* word will take effect at the next tool change.

Note

When LinuxCNC is configured for a nonrandom toolchanger (see the entry for `RANDOM_TOOLCHANGER` in the [EMCIO Section](#)), *T0* gets special handling: no tool will be selected. This is useful if you want the spindle to be empty after a tool change.

Note

When LinuxCNC is configured for a random toolchanger (see the entry for `RANDOM_TOOLCHANGER` in the [EMCIO Section](#)), *T0* does not get any special treatment: *T0* is a valid tool like any other. It is customary to use *T0* on a random toolchanger machine to track an empty pocket, so that it behaves like a nonrandom toolchanger machine and unloads the spindle.

It is an error if:

- a negative *T* number is used,
- *T* number is used that does not appear in the tool table file (with the exception that *T0* on nonrandom toolchangers **is** accepted, as noted above).

On some machines, the carousel will move when a *T* word is programmed, at the same time machining is occurring. On such machines, programming the *T* word several lines before a tool change will save time. A common programming practice for such machines is to put the *T* word for the next tool to be used on the line after a tool change. This maximizes the time available for the carousel to move.

Rapid moves after a *T*<*n*> will not show on the AXIS preview until after a feed move. This is for machines that travel long distances to change the tool like a lathe. This can be very confusing at first. To turn this feature off for the current tool program a *G1* without any move after the *T*<*n*>.

6.7 G-Code Examples

After you install LinuxCNC several sample files are placed in the `/nc_files` folder. Make sure the sample file is appropriate for your machine before running.

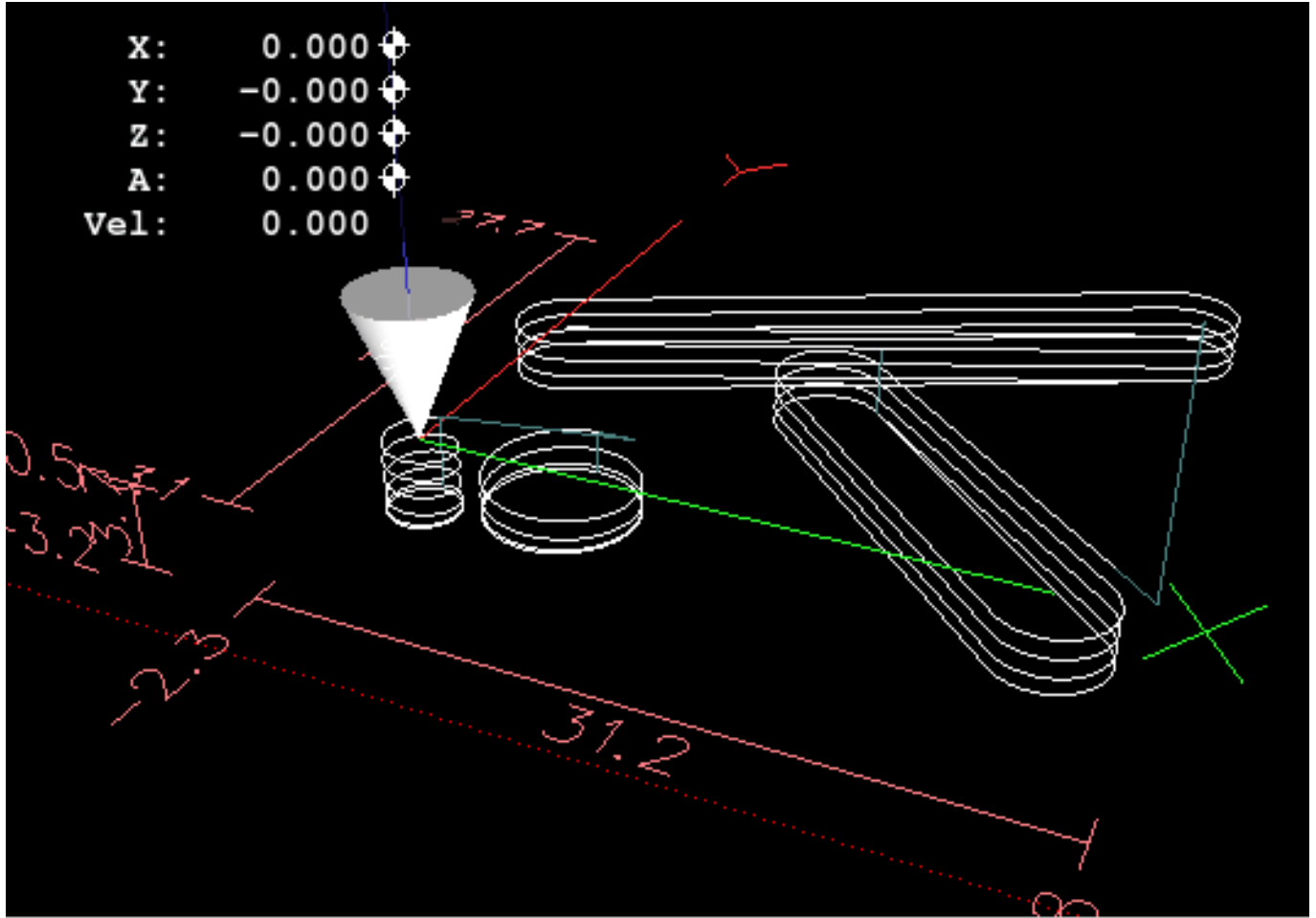
6.7.1 Mill Examples

6.7.1.1 Helical Hole Milling

- File Name: `useful-subroutines.ngc`
 - Description: Subroutine for milling a hole using parameters.
-

6.7.1.2 Slotting

- File Name: useful-subroutines.ngc
- Description: Subroutine for milling a slot using parameters.



6.7.1.3 Grid Probe

- File Name: gridprobe.ngc
- Description: Rectangular Probing

This program repeatedly probes in a regular XY grid and writes the probed location to the file *probe-results.txt* in the same directory as the .ini file.



6.7.1.4 Smart Probe

- File Name: smartprobe.ngc
- Description: Rectangular Probing

This program repeatedly probes in a regular XY grid and writes the probed location to the file *probe-results.txt* in the same directory as the .ini file. This is improved from the grid probe file.



6.7.1.5 Tool Length Probe

- File Name: tool-length-probe.ngc
- Description: Tool Length Probing

This program shows an example of how to measure tool lengths automatically using a switch hooked to the probe input. This is useful for machines without tool holders, where the length of a tool is different every time it is inserted.

6.7.1.6 Hole Probe

- File Name: probe-hole.ngc
- Description: Finding the Center and Diameter of a hole.

The program demonstrates how to find the center of a hole, measure the hole diameter and record the results.

6.7.1.7 Cutter Compensation

- File Name: comp-g1.ngc

- Description: Entry and exit movements with compensation of tool radius.

This program demonstrates the peculiarity of the toolpath without and with tool radius compensation. The tool radius is taken from the tool table.

6.7.2 Lathe Examples

6.7.2.1 Threading

- File Name lathe-g76.ngc
- Description: Facing, threading and parting off.

This file shows an example of threading on a lathe using parameters.

6.8 RS274/NGC Differences

6.8.1 Changes from RS274/NGC

Differences that change the meaning of RS274/NGC programs

Location after a tool change

In LinuxCNC, the machine does not return to its original position after a tool change. This change was made because the new tool might be longer than the old tool, and the move to the original machine position could therefore leave the tool tip too low.

Offset parameters are INI file units

In LinuxCNC, the values stored in parameters for the G28 and G30 home locations, the P1...P9 coordinate systems, and the G92 offset are in "INI file units". This change was made because otherwise the meaning of a location changed depending on whether G20 or G21 was active when G28, G30, G10 L2, or G92.3 is programmed.

Tool table lengths/diameters are in INI file units

In LinuxCNC, the tool lengths (offsets) and diameters in the tool table are specified in INI file units only. This change was made because otherwise the length of a tool and its diameter would change based on whether G20 or G21 was active when initiating G43, G41, G42 modes. This made it impossible to run G-code in the machine's non-native units, even when the G-code was simple and well-formed (starting with G20 or G21, and didn't change units throughout the program), without changing the tool table.

G84, G87 not implemented

G84 and G87 are not currently implemented, but may be added to a future release of LinuxCNC.

G28, G30 with axis words

When G28 or G30 is programmed with only some axis words present, LinuxCNC only moves the named axes. This is common on other machine controls. To move some axes to an intermediate point and then move all axes to the predefined point, write two lines of G code:

```
G0 X- Y- (axes to move to intermediate point)
G28 (move all axes to predefined point)
```

6.8.2 Additions to RS274/NGC

Differences that do not change the meaning of RS274/NGC programs

G33, G76 threading codes

These codes are not defined in RS274/NGC.

G38.2

The probe tip is not retracted after a G38.2 movement. This retraction move may be added in a future release of LinuxCNC.

G38.3...G38.5

These codes are not defined in RS274/NGC

O-codes

These codes are not defined in RS274/NGC

M50...M53 overrides

These codes are not defined in RS274/NGC

M61..M66

These codes are not defined in RS274/NGC

G43, G43.1

Negative Tool Lengths

The RS274/NGC spec says "it is expected that" all tool lengths will be positive. However, G43 works for negative tool lengths.

Lathe tools

G43 tool length compensation can offset the tool in both the X and Z dimensions. This feature is primarily useful on lathes.

Dynamic tool lengths

LinuxCNC allows specification of a computed tool length through G43.1 I K.

G41.1, G42.1

LinuxCNC allows specification of a tool diameter and, if in lathe mode, orientation in the G-code. The format is G41.1/G42.1 D L, where D is diameter and L (if specified) is the lathe tool orientation.

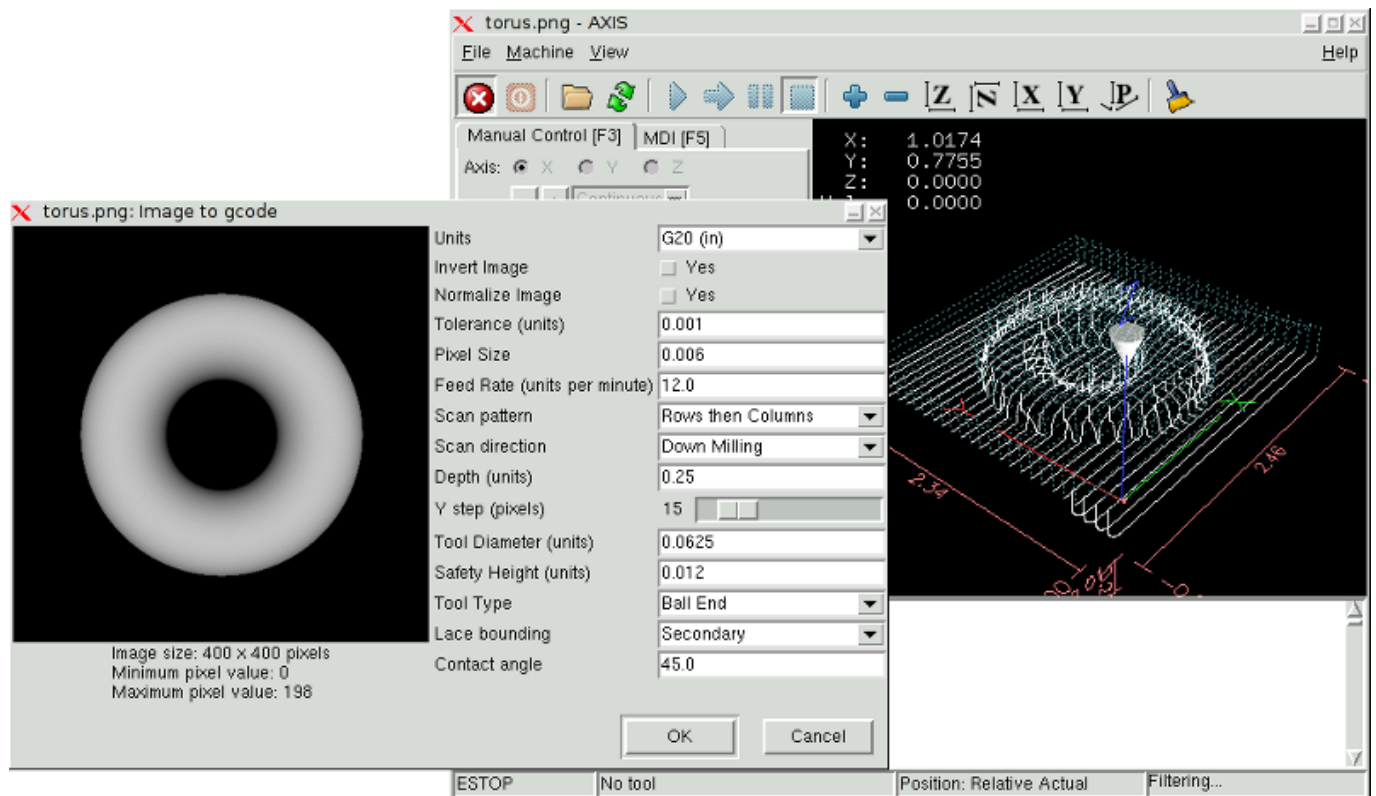
G43 without H word

In NGC, this is not allowed. In LinuxCNC, it sets length offsets for the currently loaded tool. If no tool is currently loaded, it is an error. This change was made so the user doesn't have to specify the tool number in two places for each tool change, and because it's consistent with the way G41/G42 work when the D word is not specified.

U, V, and W axes

LinuxCNC allows machines with up to 9 axes by defining an additional set of 3 linear axes known as U, V and W

6.9 Image to G-Code



6.9.1 What is a depth map?

A depth map is a greyscale image where the brightness of each pixel corresponds to the depth (or height) of the object at each point.

6.9.2 Integrating image-to-gcode with the AXIS user interface

Add the following lines to the `[FILTER]` section of your `.ini` file to make AXIS automatically invoke image-to-gcode when you open a `.png`, `.gif`, or `.jpg` image

```
PROGRAM_EXTENSION = .png,.gif,.jpg Grayscale Depth Image
png = image-to-gcode
gif = image-to-gcode
jpg = image-to-gcode
```

The standard `sim/axis.ini` configuration file is already configured this way.

6.9.3 Using image-to-gcode

Start image-to-gcode either by opening an image file in AXIS, or by invoking image-to-gcode from the terminal, as follows:

```
image-to-gcode torus.png > torus.ngc
```

Verify all the settings in the right-hand column, then press OK to create the gcode. Depending on the image size and options chosen, this may take from a few seconds to a few minutes. If you are loading the image in AXIS, the gcode will automatically be loaded and previewed once image-to-gcode completes. In AXIS, hitting reload will show the image-to-gcode option screen again, allowing you to tweak them.

6.9.4 Option Reference

6.9.4.1 Units

Specifies whether to use G20 (inches) or G21 (mm) in the generated G-code and as the units for each option labeled (*units*).

6.9.4.2 Invert Image

If "no", the black pixel is the lowest point and the white pixel is the highest point. If "yes", the black pixel is the highest point and the white pixel is the lowest point.

6.9.4.3 Normalize Image

If *yes*, the darkest pixel is remapped to black, the lightest pixel is remapped to white.

6.9.4.4 Expand Image Border

If *None*, the input image is used as-is, and details which are at the very edges of the image may be cut off. If *White* or *Black*, then a border of pixels equal to the tool diameter is added on all sides, and details which are at the very edges of the images will not be cut off.

6.9.4.5 Tolerance (units)

When a series of points are within *tolerance* of being a straight line, they are output as a straight line. Increasing tolerance can lead to better contouring performance in LinuxCNC, but can also remove or blur small details in the image.

6.9.4.6 Pixel Size (units)

One pixel in the input image will be this many units—usually this number is much smaller than 1.0. For instance, to mill a 2.5x2.5-inch object from a 400x400 image file, use a pixel size of .00625, because $2.5 / 400 = .00625$.

6.9.4.7 Plunge Feed Rate (units per minute)

The feed rate for the initial plunge movement.

6.9.4.8 Feed Rate (units per minute)

The feed rate for other parts of the path.

6.9.4.9 Spindle Speed (RPM)

The spindle speed S code that should be put into the gcode file.

6.9.4.10 Scan Pattern

Possible scan patterns are:

- Rows
- Columns
- Rows, then Columns
- Columns, then Rows

6.9.4.11 Scan Direction

Possible scan directions are:

- Positive: Start milling at a low X or Y axis value, and move towards a high X or Y axis value
- Negative: Start milling at a high X or Y axis value, and move towards a low X or Y axis value
- Alternating: Start on the same end of the X or Y axis travel that the last move ended on. This reduces the amount of traverse movements
- Up Milling: Start milling at low points, moving towards high points
- Down Milling: Start milling at high points, moving towards low points

6.9.4.12 Depth (units)

The top of material is always at $Z=0$. The deepest cut into the material is $Z=-depth$.

6.9.4.13 Step Over (pixels)

The distance between adjacent rows or columns. To find the number of pixels for a given units distance, compute $distance/pixel\ size$ and round to the nearest whole number. For example, if $pixel\ size=.006$ and the desired step over $distance=.015$, then use a Step Over of 2 or 3 pixels, because $.015/.006=2.5'$.

6.9.4.14 Tool Diameter

The diameter of the cutting part of the tool.

6.9.4.15 Safety Height

The height to move to for traverse movements. image-to-gcode always assumes the top of material is at $Z=0$.

6.9.4.16 Tool Type

The shape of the cutting part of the tool. Possible tool shapes are:

- Ball End
- Flat End
- 45 degree "vee"
- 60 degree "vee"

6.9.4.17 Lace bounding

This controls whether areas that are relatively flat along a row or column are skipped. This option only makes sense when both rows and columns are being milled. Possible bounding options are:

- None: Rows and columns are both fully milled.
- Secondary: When milling in the second direction, areas that do not strongly slope in that direction are skipped.
- Full: When milling in the first direction, areas that strongly slope in the second direction are skipped. When milling in the second direction, areas that do not strongly slope in that direction are skipped.

6.9.4.18 Contact angle

When *Lace bounding* is not *None*, slopes greater than *Contact angle* are considered to be *strong* slopes, and slopes less than that angle are considered to be weak slopes.

6.9.4.19 Roughing offset and depth per pass

Image-to-gcode can optionally perform rouging passes. The depth of successive roughing passes is given by *Roughing depth per pass*. For instance, entering 0.2 will perform the first roughing pass with a depth of 0.2, the second roughing pass with a depth of 0.4, and so on until the full Depth of the image is reached. No part of any roughing pass will cut closer than Roughing Offset to the final part. The following figure shows a tall vertical feature being milled. In this image, Roughing depth per pass is 0.2 inches and roughing offset is 0.1 inches.



Figure 6.9: Roughing passes and final pass

Chapter 7

Tool Compensation

7.1 Tool Compensation

7.1.1 Touch Off

Using the Touch Off Screen in the AXIS interface you can update the tool table automatically. Typical steps for updating the tool table:

- After homing load a tool with $Tn M6$ where n is the tool number.
- Move tool to an established point using a gauge or take a test cut and measure.
- Click the *Touch Off* button in the Manual Control tab (or hit the *End* button on your keyboard).
- Select *Tool Table* in the Coordinate System drop down box.
- Enter the gauge or measured dimension and select OK.

The Tool Table will be changed with the correct Z length to make the DRO display the correct Z position and a G43 command will be issued so the new tool Z length will be in effect. Tool table touch off is only available when a tool is loaded with $Tn M6$.

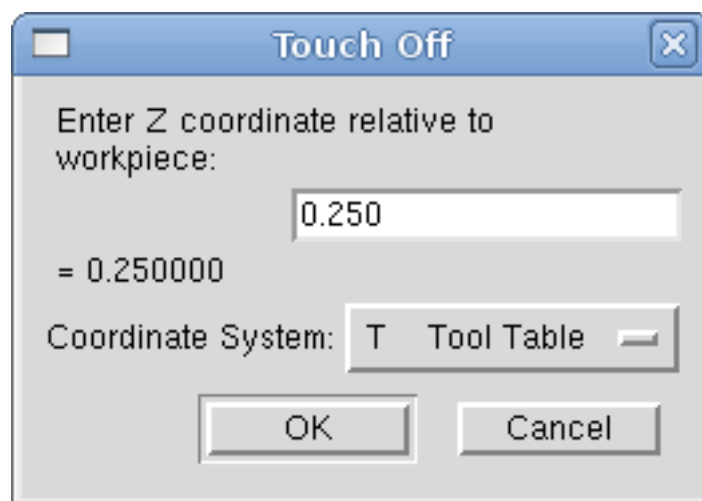


Figure 7.1: Touch Off Tool Table

7.1.1.1 Using G10 L1/L10/L11

The G10 L1/L10/L11 commands can be used to set tool table offsets:

- G10 L1 Pn - Set offset(s) to a value. Current position irrelevant. (see [G10 L1](#) for details)
- G10 L10 Pn - Set offset(s) so current position w/ fixture 1-8 becomes a value. (see [G10 L10](#) for details)
- G10 L11 Pn - Set offset(s) so current position w/ fixture 9 becomes a value. (see [G10 L11](#) for details)

Note

This is only a brief presentation, refer to the reference guide of the G-code for more detailed explanations.

7.1.2 Tool Table

The *Tool Table* is a text file that contains information about each tool. The file is located in the same directory as your configuration and is called *tool.tbl* by default. A file name may be specified with the INI file [EMCIO]TOOL_TABLE setting. The tools might be in a tool changer or just changed manually. The file can be edited with a text editor or be updated using G10 L1. See the [Lathe Tool Table](#) Section for an example of the lathe tool table format. The maximum pocket number is 1000.

The [Tool Editor](#) or a text editor can be used to edit the tool table. If you use a text editor make sure you reload the tool table in the GUI.

7.1.2.1 Tool Table Format

Table 7.1: Tool Table Format

;T#	P#	X	Y	Z	A	B	C	U	V	W	Dia	FA	BA	Ori	Rem
T1	P17	X0	Y0	Z0	A0	B0	C0	U0	V0	W0	D0	I0	J0	Q0	;rem
T2	P5	X0	Y0	Z0	A0	B0	C0	U0	V0	W0	D0	I0	J0	Q0	;rem
T3	P12	X0	Y0	Z0	A0	B0	C0	U0	V0	W0	D0	I0	J0	Q0	;rem

In general, the tool table line format is:

- T - tool number (tool numbers must be unique)
 - P - pocket number, 1-1000 (pocket numbers must be unique, Pocket 0 represents the spindle)
 - X..W - tool offset on specified axis - floating-point
 - D - tool diameter - floating-point, absolute value
 - I - front angle (lathe only) - floating-point
 - J - back angle (lathe only) - floating-point
 - Q - tool orientation (lathe only) - integer, 0-9
 - ; - beginning of comment or remark - text
-

Tool numbers should be unique. Lines beginning with a semicolon are ignored.

The units used for the length, diameter, etc., are in machine units.

You will probably want to keep the tool entries in ascending order, especially if you are going to be using a randomizing tool changer. Although the tool table does allow for tool numbers in any order.

The entries for T (tool number) and P (pocket number) are required. The last entry (a remark or comment, preceded by a semicolon) is optional. It makes reading easier if the entries are arranged in columns, as shown in the table, but the only format requirement is that there be at least one space or tab after each of the entries on a line and a newline character at the end of each entry.

The meanings of the entries and the type of data to be put in each are as follows.

Tool Number (required)

The *T* column contains the number (unsigned integer) which represents a code number for the tool. The user may use any code for any tool, as long as the codes are unsigned integers.

Pocket Number (required)

The *P* column contains the number (unsigned integer) which represents the pocket number (slot number) of the tool changer slot where the tool can be found. The entries in this column must all be different.

The pocket numbers will typically start at 1 and go up to the highest available pocket on your tool changer. But not all tool changers follow this pattern. Your pocket numbers will be determined by the numbers that your tool changer uses to refer to the pockets. So all this is to say that the pocket numbers you use will be determined by the numbering scheme used in your tool changer, and the pocket numbers you use must make sense on your machine.

Data Offset Numbers (optional)

The *Data Offset* columns (XYZABCUVW) contain real numbers which represent tool offsets in each axis. This number will be used if tool length offsets are being used and this tool is selected. These numbers can be positive, zero, or negative, and are in fact completely optional. Although you will probably want to make at least one entry here, otherwise there would be little point in making an entry in the tool table to begin with.

In a typical mill, you probably want an entry for Z (tool length offset). In a typical lathe, you probably want an entry for X (X tool offset) and Z (Z tool offset). In a typical mill using cutter diameter compensation (cutter comp), you probably also want to add an entry for D (cutter diameter). In a typical lathe using tool nose diameter compensation (tool comp), you probably also want to add an entry for D (tool nose diameter).

A lathe also requires some additional information to describe the shape and orientation of the tool. So you probably want to have entries for I (tool front angle) and J (tool back angle). You probably also want an entry for Q (tool orientation).

See the [Lathe User Information](#) chapter for more detail.

The *Diameter* column contains a real number. This number is used only if cutter compensation is turned on using this tool. If the programmed path during compensation is the edge of the material being cut, this should be a positive real number representing the measured diameter of the tool. If the programmed path during compensation is the path of a tool whose diameter is nominal, this should be a small number (positive or negative, but near zero) representing only the difference between the measured diameter of the tool and the nominal diameter. If cutter compensation is not used with a tool, it does not matter what number is in this column.

The *Comment* column may optionally be used to describe the tool. Any type of description is OK. This column is for the benefit of human readers only. The comment must be preceded by a semicolon.

Note

Earlier versions of LinuxCNC had two different tool table formats for mills and lathes, but since the 2.4.x release, one tool table format is used for all machines.

7.1.2.2 Tool IO

The userspace program specified by **[EMCIO]EMCIO = io** is conventionally used for tool changer management (and other io functions for enabling LinuxCNC and the control of coolant/lube hardware). The HAL pins used for tool management are prefixed with **iocontrol.0..**

A G-code **T** command asserts the HAL output pin **iocontrol.0.tool-prepare**. The HAL input pin, **iocontrol.0.tool-prepared**, must be set by external HAL logic to complete tool preparation leading to a subsequent reset of the tool-prepare pin.

A G-code **M6** command asserts the HAL output pin **iocontrol.0.tool-change**. The related HAL input pin, **iocontrol.0.tool-prepared**, must be set by external HAL logic to indicate completion of the tool change leading to a subsequent reset of the tool-change pin.

Tooldata is accessed by an ordered index (idx) that depends on the type of toolchanger specified by **[EMCIO]RANDOM_TOOLCHANGER=type**.

1. For **RANDOM_TOOLCHANGER = 0**, (0 is default and specifies a non-random toolchanger) idx is a number indicating the sequence in which tooldata was loaded.
2. For **RANDOM_TOOLCHANGER = 1**, idx is the **current** pocket number for the toolnumber specified by the G-code select tool command **Tn**.

The io program provides HAL output pins to facilitate toolchanger management:

1. **iocontrol.0.tool-prep-number**
2. **iocontrol.0.tool-prep-index**
3. **iocontrol.0.tool-prep-pocket**
4. **iocontrol.0.tool-from-pocket**

1. Tool number $n == 0$ indicates no tool
2. The pocket number for a tool is set when tooldata is loaded/reloaded from its data source ([EMCIO]TOOL_TABLE or [EMCIO]DB_PROGRAM).
3. At G-code **Tn** ($n \neq 0$) command:
 - a. **iocontrol.0.tool-prep-index** = idx (index based on tooldata load sequence)
 - b. **iocontrol.0.tool-prep-number** = n
 - c. **iocontrol.0.tool-prep-pocket** = the pocketno for n
4. At G-code **T0** ($n == 0$ remove) command:
 - a. **iocontrol.0.tool-prep-index** = 0
 - b. **iocontrol.0.tool-prep-number** = 0
 - c. **iocontrol.0.tool-prep-pocket** = 0
5. At M-code **M6** (following iocontrol.0.tool-changed pin 0-->1):
 - a. **iocontrol.0.tool-from-pocket** = pocket number used to retrieve tool

1. Tool number $n == 0$ is **not special**.
2. Pocket number 0 is **special** as it indicates the **spindle**.
3. The **current** pocket number for tool N is the tooldata index (idx) for tool n .

4. At G-code command **Tn**:

- a. **iocontrol.0.tool-prep-index** = tooldata index (idx) for tool n
- b. **iocontrol.0.tool-prep-number** = n
- c. **iocontrol.0.tool-prep-pocket** = pocket number for tool n

5. At M-code **M6** (following iocontrol.0.tool-changed pin 0-->1):

- a. **iocontrol.0.tool-from-pocket** = pocket number used to retrieve tool

Note

At startup, **iocontrol.0.tool-from-pocket** = 0. An M61Qn (n!=0) command does not change the **iocontrol.0.tool-from-pocket**. An M61Qn (n==0) command sets **iocontrol.0.tool-from-pocket** to 0.

7.1.2.3 Tool Changers

LinuxCNC supports three types of tool changers: *manual*, *random location* and *non-random or fixed location*. Information about configuring a LinuxCNC tool changer is in the [EMCIO Section](#) of the INI chapter.

Manual Tool Changer Manual tool changer (you change the tool by hand) is treated like a fixed location tool changer. Manual toolchanges can be aided by a HAL configuration that employs the userspace program **hal_manualtoolchange** and is typically specified in an INI file with INI statements:

```
[HAL]
HALFILE = axis_manualtoolchange.hal
```

Fixed Location Tool Changers Fixed location tool changers always return the tools to a fixed position in the tool changer. This would also include designs like lathe turrets. When LinuxCNC is configured for a fixed location tool changer the *P* number is not used internally (but read, preserved and rewritten) by LinuxCNC, so you can use *P* for any bookkeeping number you want.

Note

When using [EMCIO]RANDOM_TOOLCHANGER = 0 (the default), the *P* pocket number is a parameter of the tooldata that is retrieved from the tooldata source ([EMCIO]TOOL_TABLE or [EMCIO]DB_PROGRAM). In many applications it is fixed but it may be changed by edits to the [EMCIO]TOOL_TABLE or programatically when the [EMCIO]DB_PROGRAM is used. LinuxCNC pushes updates to the data source ([EMCIO]TOOL_TABLE or [EMCIO]DB_PROGRAM) for gcodes G10L1,G10L10,G10L11,M61. LinuxCNC can pull tooldata updates from the data source by UI (user-interface) commands (python example: linuxcnc.command().load_tool_table()) or by the gcode: G10L0.

Random Location Tool Changers Random location tool changers ([EMCIO]RANDOM_TOOLCHANGER = 1) swap the tool in the spindle with the one in the changer. With this type of tool changer the tool will always be in a different pocket after a tool change. When a tool is changed LinuxCNC rewrites the pocket number to keep track of where the tools are. *T* can be any number but *P* must be a number that makes sense for the machine.

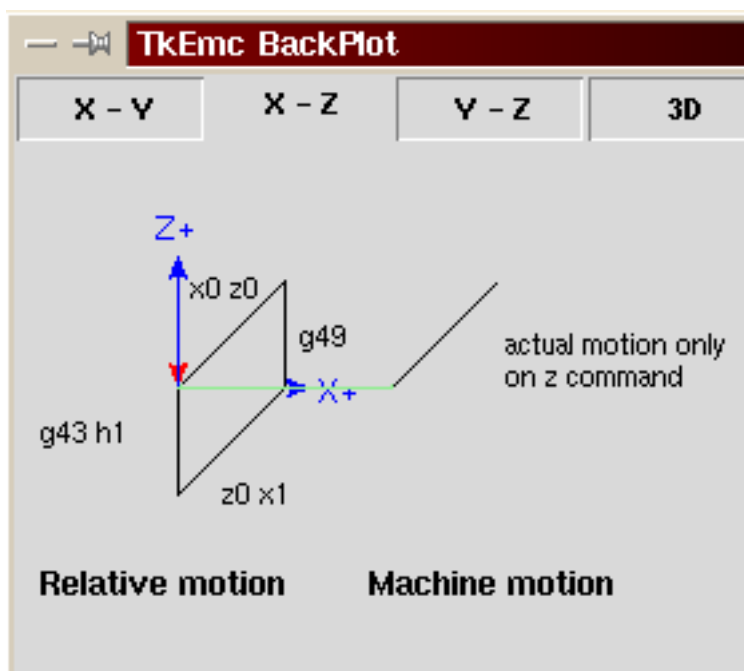
7.1.3 Tool Length Compensation

The tool length compensations are given as positive numbers in the tool table. A tool compensation is programmed using G43 H'n', where n is the index number of the desired tool in the tool table. It is intended that all entries in the tool table are positive. The value of H is checked, it must be a non-negative integer when read. The interpreter behaves as follows:

1. If G43 H'n' is programmed, a call to the function `USE_TOOL_LENGTH_OFFSET(length)` is made (where length is the length difference, read from the tool table, of the indexed tool n), `tool_length_offset` is repositioned in the machine settings model and the value of `current_z` in the model is adjusted. Note that n does not have to be the same as the slot number of the tool currently in the spindle.
2. If G49 is programmed, `USE_TOOL_LENGTH_OFFSET(0.0)` is called, `tool_length_offset` is reset to 0.0 in the machine settings template and the current value of `current_z` in the model is adjusted. The effect of the tool length compensation is illustrated in the capture below. Note that the tool length is subtracted from Z so that the programmed control point corresponds to the tip of the tool. Note also that the effect of the length compensation is immediate when you see the compensation is immediate when the position of Z is seen as a relative coordinate, but it has no effect on the actual machine position until a Z movement is programmed.

Tool length test program. Tool #1 is one inch long.

```
N01 G1 F15 X0 Y0 Z0
N02 G43 H1 Z0 X1
N03 G49 X0 Z0
N04 G0 X2
N05 G1 G43 H1 G4 P10 Z0 X3
N06 G49 X2 Z0
N07 G0 X0
```



With this program, in most cases, the machine will apply the offset in the form of a ramp during the movement in xyz following the word G43.

7.1.4 Cutter Radius Compensation

Cutter Compensation allows the programmer to program the tool path without knowing the exact tool diameter. The only caveat is the programmer must program the lead in move to be at least as long as

the largest tool radius that might be used.

There are two possible paths the cutter can take since the cutter compensation can be on to the left or right side of a line when facing the direction of cutter motion from behind the cutter. To visualize this imagine you were standing on the part walking behind the tool as it progresses across the part. G41 is your left side of the line and G42 is the right side of the line.

The end point of each move depends on the next move. If the next move creates an outside corner the move will be to the end point of the compensated cut line. If the next move creates in an inside corner the move will stop short so to not gouge the part. The following figure shows how the compensated move will stop at different points depending on the next move.

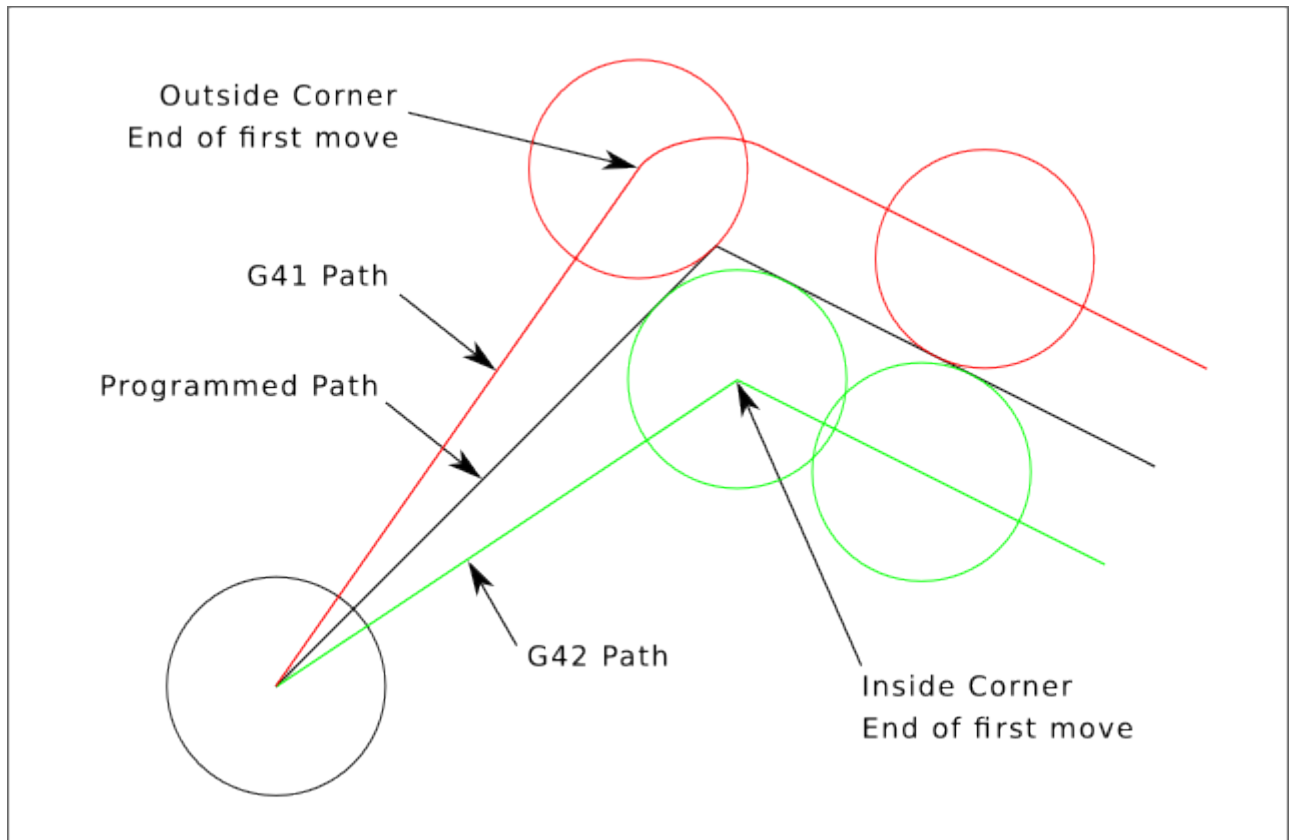


Figure 7.2: Compensation End Point

7.1.4.1 Overview

Cutter compensation uses the data from the tool table to determine the offset needed. The data can be set at run time with G10 L1.

Any move that is long enough to perform the compensation will work as the entry move. The minimum length is the cutter radius. This can be a rapid move above the work piece. If several rapid moves are issued after a G41/42 only the last one will move the tool to the compensated position.

In the following figure you can see that the entry move is compensated to the right of the line. This puts the center of the tool to the right of X0 in this case. If you were to program a profile and the end is at X0 the resulting profile would leave a bump due to the offset of the entry move.

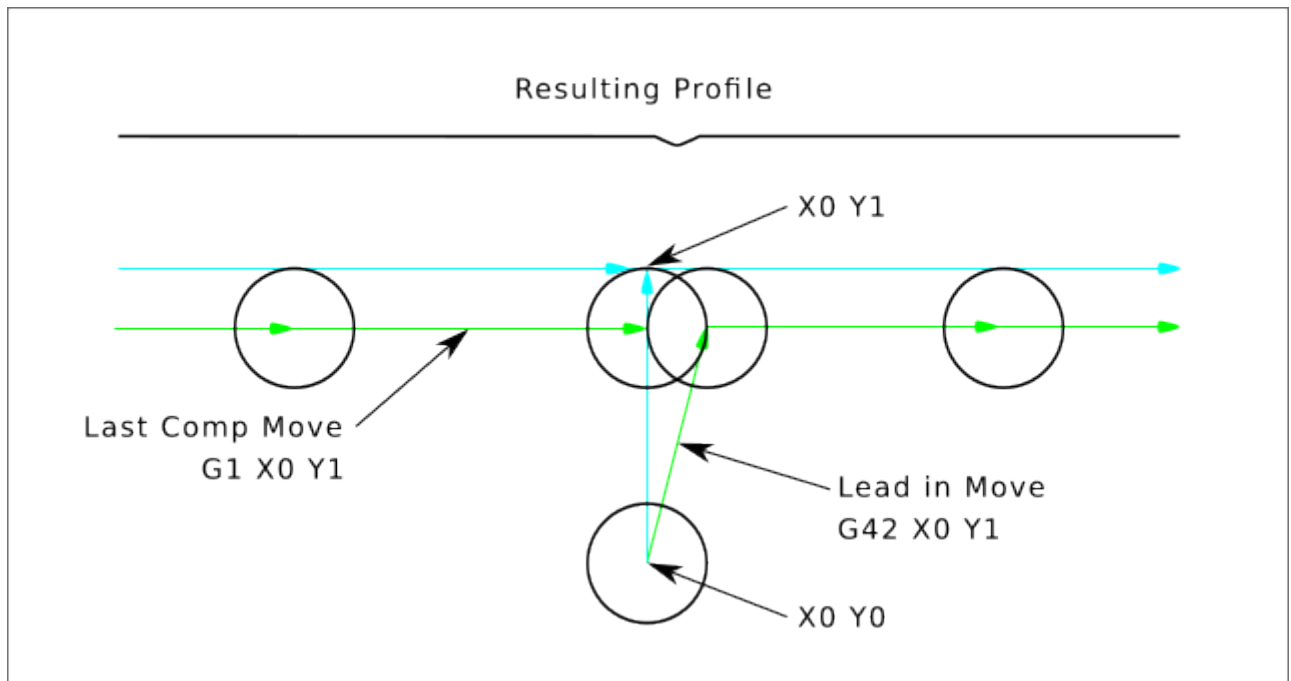


Figure 7.3: Entry Move

Z axis motion may take place while the contour is being followed in the XY plane. Portions of the contour may be skipped by retracting the Z axis above the part and by extending the Z-axis at the next start point.

Rapid moves may be programmed while compensation is turned on.

Start a program with G40 to make sure compensation is off.

7.1.4.2 Examples

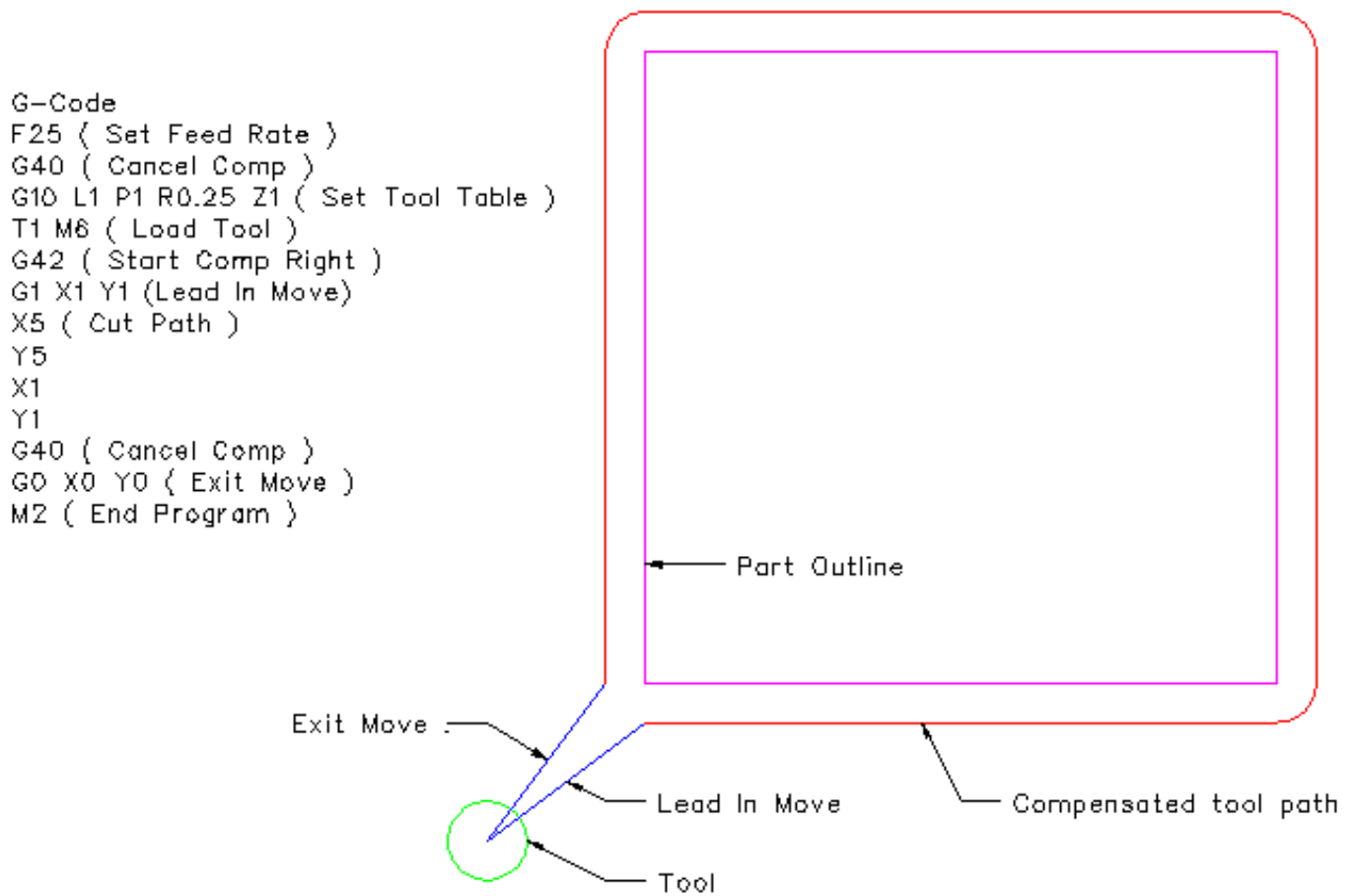


Figure 7.4: Outside Profile

```

G20 ( Inch Mode )
F30 ( Set Feed Rate )
G10 L1 P1 R.25 Z1 ( Set Tool Table )
T1 M6 ( Load the Tool )
G0 Z0 ( Move to safe Z height )
G41 ( Start Cutter Comp Left )
X4 Y3 ( Rapid to start point )
G1 X5 Z-1 ( Move to cut height )
G3 X6 Y4 J1 ( Arc into cut path )
G1 Y6 ( Cut Profile )
X2
Y2
X6
Y4
G3 X5 Y5 I-1 ( Arc out of cut path )
G0 Z0 ( Move cutter to safe Z height )
G40 ( Stop Cutter Comp )
G0 X1 Y1 ( Move to safe position )
T0 M6 ( Remove Tool )
M2 ( End Program )

```

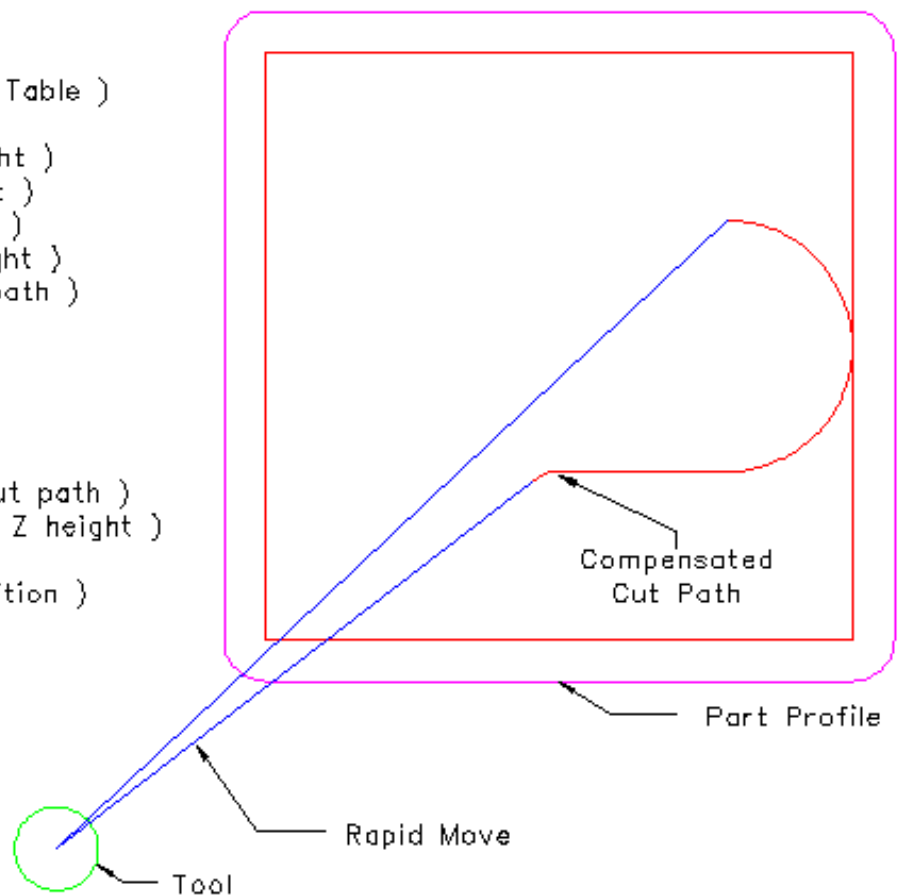


Figure 7.5: Inside Profile

7.2 Tool Edit GUI

7.2.1 Overview

Note

The tooledit elements described here are available in version 2.5.1 and later. In version 2.5.0, the graphical interface interface does not allow these adjustments.



Figure 7.6: Tool Edit GUI - Overview

The *tooledit* program can update the tool table file with edited changes by using the SaveFile button. The SaveFile button updates the system file but a separate action is required to update the tool table data used by a running LinuxCNC instance. With the axis GUI, both the file and the current tool table data used by LinuxCNC can be updated with the ReloadTable button. This button is enabled only when the machine is ON and IDLE.

7.2.2 Column Sorting

The tool table display can be sorted on any column in ascending order by clicking on the column header. A second click sorts in descending order. Column sorting requires that the machine is configured with the default tcl version ≥ 8.5 .



Figure 7.7: Tool Edit GUI - Column Sorting

In Ubuntu Lucid 10.04 tcl/tk8.4 is installed by default. The installation is performed as follows:

```
sudo apt-get install tcl8.5 tk8.5
```

Depending upon other applications installed on the system, it may be necessary to enable tcl/tk8.5 with the commands:

```
sudo update-alternatives --config tclsh ;# select the option for tclsh8.5
sudo update-alternatives --config wish ;# select the option for wish8.5
```

7.2.3 Columns Selection

By default, the *tooledit* program will display all possible tool table parameter columns. Since few machines use all parameters, the columns displayed can be limited with the following ini file setting:

Syntax of .INI file

```
[DISPLAY]
TOOL_EDITOR = tooledit column_name column_name ...
```

Example for Z and DIAM columns

```
[DISPLAY]
TOOL_EDITOR = tooledit Z DIAM
```

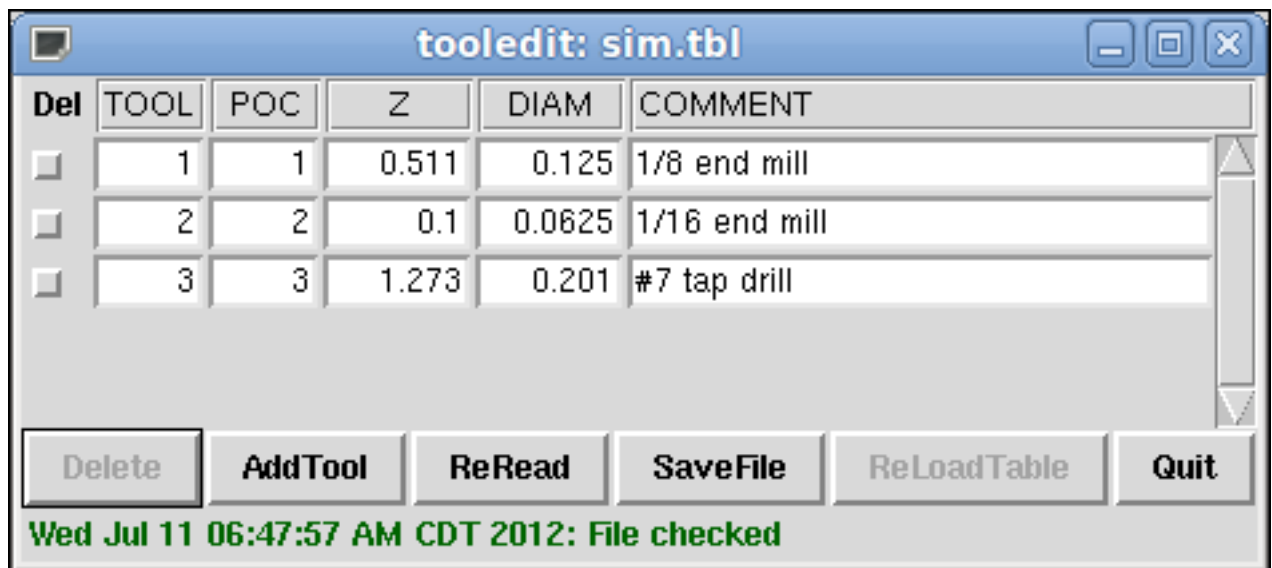


Figure 7.8: Tool Edit GUI - Columns Selection Example

7.2.4 Stand Alone Use

The *tooledit* program can also be invoked as a standalone program. For example, if the program is in the user PATH, typing *tooledit* will show the usage syntax:

Stand Alone

```
tooledit
Usage:
    tooledit filename
    tooledit [column_1 ... column_n] filename
```

Valid column names are: x y z a b c u v w diam front back orien

To synchronize a standalone *tooleedit* with a running LinuxCNC application, the filename must resolve to the same [EMCIO]TOOL_TABLE filename specified in the LinuxCNC ini file.

When using the program *tooleedit* while LinuxCNC is running, gcode command execution or other programs may alter tool table data and the tool table file. File changes are detected by *tooleedit* and a message is displayed:

```
Warning: File changed by another process
```

The *tooleedit* tool table display can be updated to read the modified file with the ReRead button.

The tool table is specified in the ini file with an entry:

```
[EMCIO]TOOL_TABLE = tool_table_filename
```

The tool table file can be edited with any simple text editor (not a word processor).

The axis GUI can optionally use an ini file setting to specify the tool editor program:

```
[DISPLAY]TOOL_EDITOR = path_to_editor_program
```

By default, the program named *tooleedit* is used. This editor supports all tool table parameters, allows addition and deletion of tool entries, and performs a number of validity checks on parameter values.

Part IV

Configuration

Chapter 8

General Info

8.1 Integrator Concepts

8.1.1 File Locations

LinuxCNC looks for the configuration and G-code files in a specific place. The location depends on how you run LinuxCNC.

8.1.1.1 Installed

If your running LinuxCNC from the LiveCD or you installed via a .deb and use the configuration picker *LinuxCNC* from the menu LinuxCNC looks in the following directories:

- The LinuxCNC directory is located at */home/user-name/linuxcnc*.
- The Configuration directories are located at */home/user-name/linuxcnc/configs*.
 - Configuration files are located at */home/user-name/linuxcnc/configs/name-of-config*.
- G-code files are located at */home/user-name/linuxcnc/nc_files*'.

For example for a configuration called Mill and a user name Fred the directory and file structure would look like this.

- */home/fred/linuxcnc*
 - */home/fred/linuxcnc/nc_files*
 - */home/fred/linuxcnc/configs/mill*
 - */home/fred/linuxcnc/configs/mill/mill.ini*
 - */home/fred/linuxcnc/configs/mill/mill.hal*
 - */home/fred/linuxcnc/configs/mill/mill.var*
 - */home/fred/linuxcnc/configs/mill/tool.tbl*
-

8.1.1.2 Command Line

If you run LinuxCNC from the command line and specify the name and location of the INI file the file locations can be in a different place. To view the options for running LinuxCNC from the command line run *linuxcnc -h*.

Note

Optional locations for some files can be configured in the INI file. See the `<<sub:ini:sec:display,[DISPLAY]>>` section and the `<<sub:ini:sec:rs274ngc,[RS274NGC]>>` section.

8.1.2 Files

Each configuration directory requires at least the following files:

- An INI file `.ini`
- A HAL file `.hal` or HALTCL file `.tcl` specified in the [HAL](#) section of the INI file.

Note

Other files may be required for some GUIs.

Optionally you can also have:

- A Variables file `.var`
 - If you omit a `.var` file in a directory but include `<<sub:ini:sec:rs274ngc,[RS274NGC]>> PARAMETER_FILE=somefilename.var`, the file will be created for you when LinuxCNC starts.
 - If you omit a `.var` file and omit the item `[RS274NGC] PARAMETER_FILE`, a var file named `rs274ngc.var` will be created when LinuxCNC starts. There may be some confusing messages if `[RS274NGC]PARAMETER_FILE` is omitted.
- A Tool Table file `.tbl` if `<<sub:ini:sec:emcmot,[EMCMOT]>> TOOL_TABLE` has been specified in the INI file. Some configurations do not need a tool table.

8.1.3 Stepper Systems

8.1.3.1 Base Period

`BASE_PERIOD` is the *heartbeat* of your LinuxCNC computer.¹ Every period, the software step generator decides if it is time for another step pulse. A shorter period will allow you to generate more pulses per second, within limits. But if you go too short, your computer will spend so much time generating step pulses that everything else will slow to a crawl, or maybe even lock up. Latency and stepper drive requirements affect the shortest period you can use.

Worst case latencies might only happen a few times a minute, and the odds of bad latency happening just as the motor is changing direction are low. So you can get very rare errors that ruin a part every once in a while and are impossible to troubleshoot.

¹This section refers to using **stepgen**, LinuxCNC's built-in step generator. Some hardware devices have their own step generator and do not use LinuxCNC's built-in one. In that case, refer to your hardware manual.

The simplest way to avoid this problem is to choose a `BASE_PERIOD` that is the sum of the longest timing requirement of your drive, and the worst case latency of your computer. This is not always the best choice. For example, if you are running a drive with a 20 μs direction signal hold time requirement, and your latency test said you have a maximum latency of 11 μs , then if you set the `BASE_PERIOD` to $20+11 = 31 \mu\text{s}$ you get a not-so-nice 32,258 steps per second in one mode and 16,129 steps per second in another mode.

The problem is with the 20 μs hold time requirement. That plus the 11 μs latency is what forces us to use a slow 31 μs period. But the LinuxCNC software step generator has some parameters that let you increase the various times from one period to several. For example, if *steplen* ² is changed from 1 to 2, then there will be two periods between the beginning and end of the step pulse. Likewise, if *dirhold* ³ is changed from 1 to 3, there will be at least three periods between the step pulse and a change of the direction pin.

If we can use *dirhold* to meet the 20 μs hold time requirement, then the next longest time is the 4.5 μs high time. Add the 11 μs latency to the 4.5 μs high time, and you get a minimum period of 15.5 μs . When you try 15.5 μs , you find that the computer is sluggish, so you settle on 16 μs . If we leave *dirhold* at 1 (the default), then the minimum time between step and direction is the 16 μs period minus the 11 μs latency = 5 μs , which is not enough. We need another 15 μs . Since the period is 16 μs , we need one more period. So we change *dirhold* from 1 to 2. Now the minimum time from the end of the step pulse to the changing direction pin is $5+16=21 \mu\text{s}$, and we don't have to worry about the drive stepping the wrong direction because of latency.

For more information on stepgen see the [stepgen section](#).

8.1.3.2 Step Timing

Step Timing and Step Space on some drives are different. In this case the Step point becomes important. If the drive steps on the falling edge then the output pin should be inverted.

8.1.4 Servo Systems

8.1.4.1 Basic Operation

Servo systems are capable of greater speed and accuracy than equivalent stepper systems, but are more costly and complex. Unlike stepper systems, servo systems require some type of position feedback device, and must be adjusted or *tuned*, as they don't quite work right out of the box as a stepper system might. These differences exist because servos are a *closed loop* system, unlike stepper motors which are generally run *open loop*. What does *closed loop* mean? Let's look at a simplified diagram of how a servomotor system is connected.

²steplen refers to a parameter that adjusts the performance of LinuxCNC's built-in step generator, *stepgen*, which is a HAL component. This parameter adjusts the length of the step pulse itself. Keep reading, all will be explained eventually.

³dirhold refers to a parameter that adjusts the length of the direction hold time.

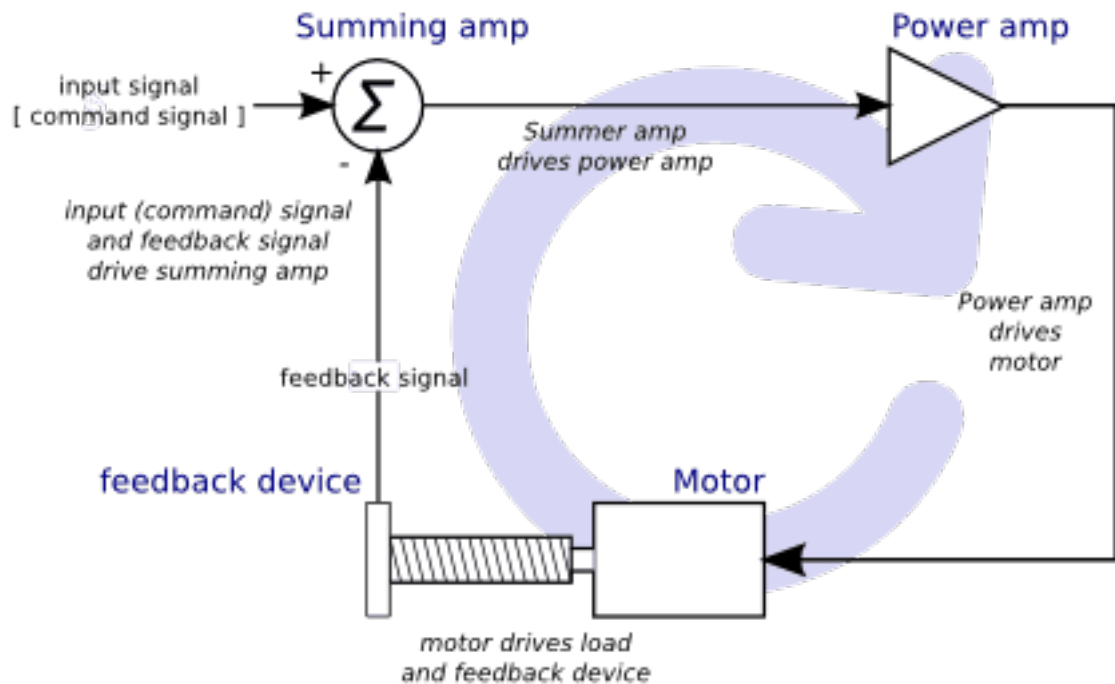


Figure 8.1: Servo Loop

This diagram shows that the input signal (and the feedback signal) drive the summing amplifier, the summing amplifier drives the power amplifier, the power amplifier drives the motor, the motor drives the load (and the feedback device), and the feedback device (and the input signal) drive the motor. This looks very much like a circle (a closed loop) where A controls B, B controls C, C controls D, and D controls A.

If you have not worked with servo systems before, this will no doubt seem a very strange idea at first, especially as compared to more normal electronic circuits, where the inputs proceed smoothly to the outputs, and never go back.⁴ If *everything* controls *everything else*, how can that ever work, who's in charge? The answer is that LinuxCNC *can* control this system, but it has to do it by choosing one of several control methods. The control method that LinuxCNC uses, one of the simplest and best, is called PID.

PID stands for Proportional, Integral, and Derivative. The Proportional value determines the reaction to the current error, the Integral value determines the reaction based on the sum of recent errors, and the Derivative value determines the reaction based on the rate at which the error has been changing. They are three common mathematical techniques that are applied to the task of getting a working process to follow a set point. In the case of LinuxCNC the process we want to control is actual axis position and the set point is the commanded axis position.

⁴If it helps, the closest equivalent to this in the digital world are *state machines*, *sequential machines* and so forth, where what the outputs are doing *now* depends on what the inputs (and the outputs) were doing *before*. If it doesn't help, then nevermind.



Figure 8.2: PID Loop

By *tuning* the three constants in the PID controller algorithm, the controller can provide control action designed for specific process requirements. The response of the controller can be described in terms of the responsiveness of the controller to an error, the degree to which the controller overshoots the set point and the degree of system oscillation.

8.1.4.2 Proportional term

The proportional term (sometimes called gain) makes a change to the output that is proportional to the current error value. A high proportional gain results in a large change in the output for a given change in the error. If the proportional gain is too high, the system can become unstable. In contrast, a small gain results in a small output response to a large input error. If the proportional gain is too low, the control action may be too small when responding to system disturbances.

In the absence of disturbances, pure proportional control will not settle at its target value, but will retain a steady state error that is a function of the proportional gain and the process gain. Despite the steady-state offset, both tuning theory and industrial practice indicate that it is the proportional term that should contribute the bulk of the output change.

8.1.4.3 Integral term

The contribution from the integral term (sometimes called reset) is proportional to both the magnitude of the error and the duration of the error. Summing the instantaneous error over time (integrating the error) gives the accumulated offset that should have been corrected previously. The accumulated error is then multiplied by the integral gain and added to the controller output.

The integral term (when added to the proportional term) accelerates the movement of the process towards set point and eliminates the residual steady-state error that occurs with a proportional only controller. However, since the integral term is responding to accumulated errors from the past, it can cause the present value to overshoot the set point value (cross over the set point and then create a deviation in the other direction).

8.1.4.4 Derivative term

The rate of change of the process error is calculated by determining the slope of the error over time (i.e., its first derivative with respect to time) and multiplying this rate of change by the derivative gain.

The derivative term slows the rate of change of the controller output and this effect is most noticeable close to the controller set point. Hence, derivative control is used to reduce the magnitude of the overshoot produced by the integral component and improve the combined controller-process stability.

8.1.4.5 Loop tuning

If the PID controller parameters (the gains of the proportional, integral and derivative terms) are chosen incorrectly, the controlled process input can be unstable, i.e., its output diverges, with or without oscillation, and is limited only by saturation or mechanical breakage. Tuning a control loop is the adjustment of its control parameters (gain/proportional band, integral gain/reset, derivative gain/rate) to the optimum values for the desired control response.

8.1.4.6 Manual tuning

A simple tuning method is to first set the I and D values to zero. Increase the P until the output of the loop oscillates, then the P should be set to be approximately half of that value for a *quarter amplitude decay* type response. Then increase I until any offset is correct in sufficient time for the process. However, too much I will cause instability. Finally, increase D, if required, until the loop is acceptably quick to reach its reference after a load disturbance. However, too much D will cause excessive response and overshoot. A fast PID loop tuning usually overshoots slightly to reach the set point more quickly; however, some systems cannot accept overshoot, in which case an *over-damped* closed-loop system is required, which will require a P setting significantly less than half that of the P setting causing oscillation.

8.1.5 RTAI

The Real Time Application Interface (RTAI) is used to provide the best Real Time (RT) performance. The RTAI patched kernel lets you write applications with strict timing constraints. RTAI gives you the ability to have things like software step generation which require precise timing.

8.1.5.1 ACPI

The Advanced Configuration and Power Interface (ACPI) has a lot of different functions, most of which interfere with RT performance (for example: power management, CPU power down, CPU frequency scaling, etc). The LinuxCNC kernel (and probably all RTAI-patched kernels) has ACPI disabled. ACPI also takes care of powering down the system after a shutdown has been started, and that's why you might need to push the power button to completely turn off your computer. The RTAI group has been improving this in recent releases, so your LinuxCNC system may shut off by itself after all.

8.2 Latency Testing

8.2.1 Latency Test

This test is the first test that should be performed on a PC to see if it is able to drive a CNC machine.

Latency is how long it takes the PC to stop what it is doing and respond to an external request. For LinuxCNC the request is BASE_THREAD that makes the periodic *heartbeat* that serves as a timing reference for the step pulses. The lower the latency, the faster you can run the heartbeat, and the faster and smoother the step pulses will be.

Latency is far more important than CPU speed. A lowly Pentium II that responds to interrupts within 10 microseconds each and every time can give better results than the latest and fastest P4 Hyper-threading beast.

The CPU isn't the only factor in determining latency. Motherboards, video cards, USB ports, and a number of other things can hurt the latency. The best way to find out what you are dealing with is to run the RTAI latency test.

Generating step pulses in software has one very big advantage - it's free. Just about every PC has a parallel port that is capable of outputting step pulses that are generated by the software. However, software step pulses also have some disadvantages:

- limited maximum step rate
- jitter in the generated pulses
- loads the CPU

The best way to find out how well your PC will run LinuxCNC is to run the HAL latency test. To run the test, open a terminal window (in Ubuntu, from Applications → Accessories → Terminal) and run the following command:

```
latency-test
```

This will start the latency test with a base-thread period of 25uS and a servo-thread period of 1mS. The period times may be specified on the command line:

```
latency-test 50000 1000000
```

This will start the latency test with a base-thread period of 50uS and a servo-thread period of 1mS. For available options, on the command line enter:

```
latency-test -h
```

After stating a latency test you should see something like this:

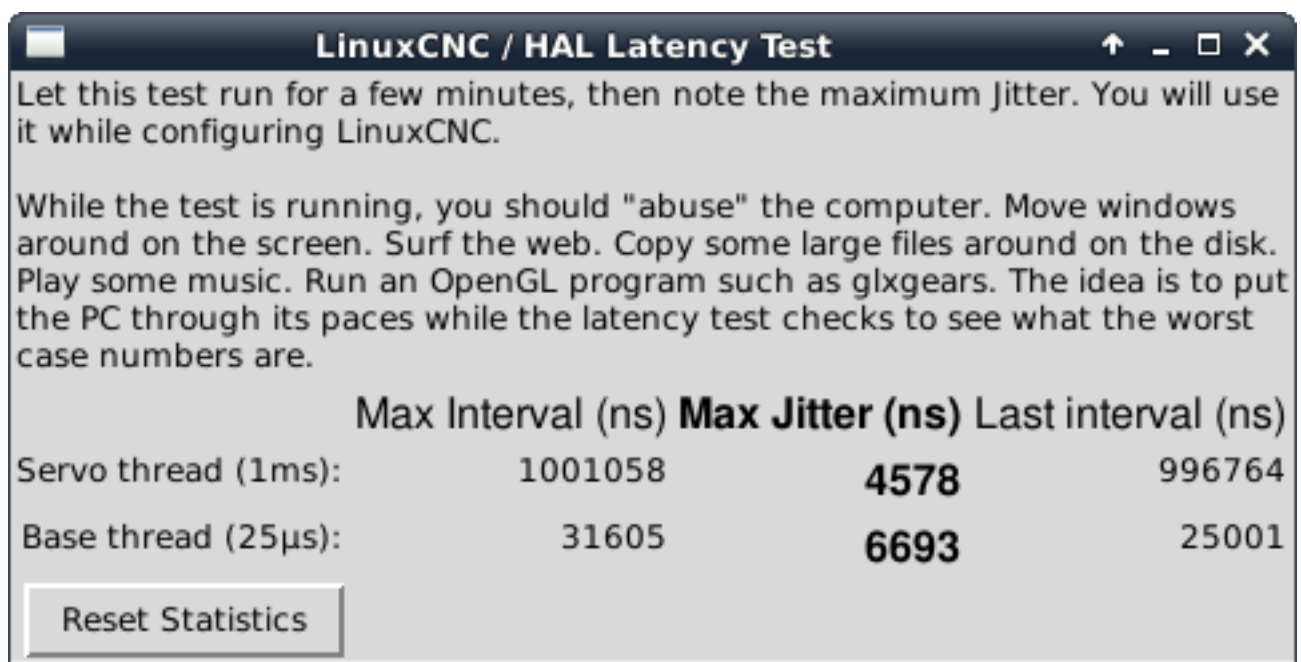


Figure 8.3: HAL Latency Test

While the test is running, you should *abuse* the computer. Move windows around on the screen. Surf the web. Copy some large files around on the disk. Play some music. Run an OpenGL program such as glxgears. The idea is to put the PC through its paces while the latency test checks to see what the worst case numbers are.

Note

Do not run LinuxCNC or Stepconf while the latency test is running.

The important number for software stepping is the *max jitter* of the base thread. In the example above, that is 6693 nanoseconds, or 6.693 microseconds. Record this number, and enter it in Stepconf when it is requested.

In the example above, latency-test only ran for a few seconds. You should run the test for at least several minutes; sometimes the worst case latency doesn't happen very often, or only happens when you do some particular action. For instance, one Intel motherboard worked pretty well most of the time, but every 64 seconds it had a very bad 300 us latency. Fortunately that was fixable, see <http://wiki.linuxcnc.org/cgi-bin/wiki.pl?FixingSMIIssues>

So, what do the results mean? If your Max Jitter number is less than about 15-20 microseconds (15000-20000 nanoseconds), the computer should give very nice results with software stepping. If the max latency is more like 30-50 microseconds, you can still get good results, but your maximum step rate might be a little disappointing, especially if you use microstepping or have very fine pitch leadscrews. If the numbers are 100 us or more (100,000 nanoseconds), then the PC is not a good candidate for software stepping. Numbers over 1 millisecond (1,000,000 nanoseconds) mean the PC is not a good candidate for LinuxCNC, regardless of whether you use software stepping or not.

Note

If you get high numbers, there may be ways to improve them. Another PC had very bad latency (several milliseconds) when using the onboard video. But a \$5 used video card solved the problem. LinuxCNC does not require bleeding edge hardware.

For more information on stepper tuning see the [Stepper Tuning](#) Chapter.

Additional command line tools are available for examining latency when LinuxCNC is not running.

8.2.2 Latency Plot

latency-plot makes a strip chart recording for a base and a servo thread. It may be useful to see spikes in latency when other applications are started or used. Usage:

```
latency-plot --help
```

Usage:

```
latency-plot --help | -?  
latency-plot --hal [Options]
```

Options:

```
--base nS  (base thread interval, default: 25000)  
--servo nS (servo thread interval, default: 1000000)  
--time mS  (report interval, default: 1000)  
--relative (relative clock time (default))  
--actual   (actual clock time)
```

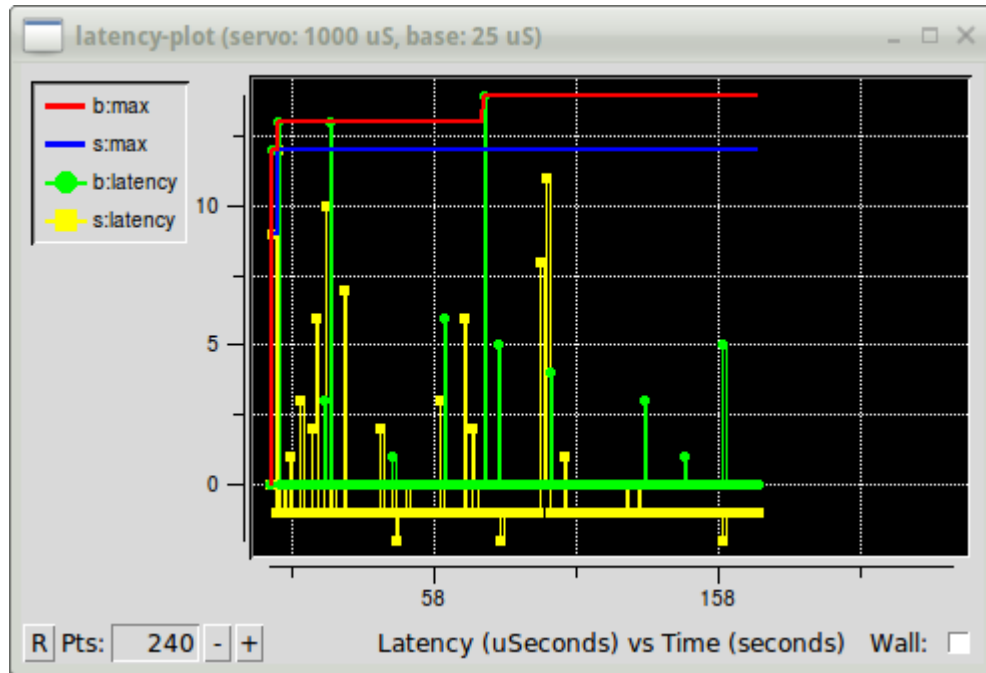


Figure 8.4: latency-plot Window

8.2.3 Latency Histogram

latency-histogram displays a histogram of latency (jitter) for a base and servo thread.

Usage:

```
latency-histogram --help | -?
latency-histogram [Options]
```

Options:

```
--base      nS   (base thread interval, default: 25000, min: 5000)
--servo     nS   (servo thread interval, default: 1000000, min: 25000)
--bbinsize  nS   (base bin size, default: 100)
--sbinsize  nS   (servo bin size, default: 100)
--bbins     n    (base bins, default: 200)
--sbins     n    (servo bins, default: 200)
--logscale  0|1  (y axis log scale, default: 1)
--text      note (additional note, default: "" )
--show      (show count of undisplayed bins)
--nobase    (servo thread only)
--verbose   (progress and debug)
--nox       (no gui, display elapsed,min,max,sdev for each thread)
```

Notes:

Linuxcnc and Hal should not be running, stop with halrun -U.
 Large number of bins and/or small binsizes will slow updates.
 For single thread, specify --nobase (and options for servo thread).
 Measured latencies outside of the +/- bin range are reported
 with special end bars. Use --show to show count for
 the off-chart [pos|neg] bin

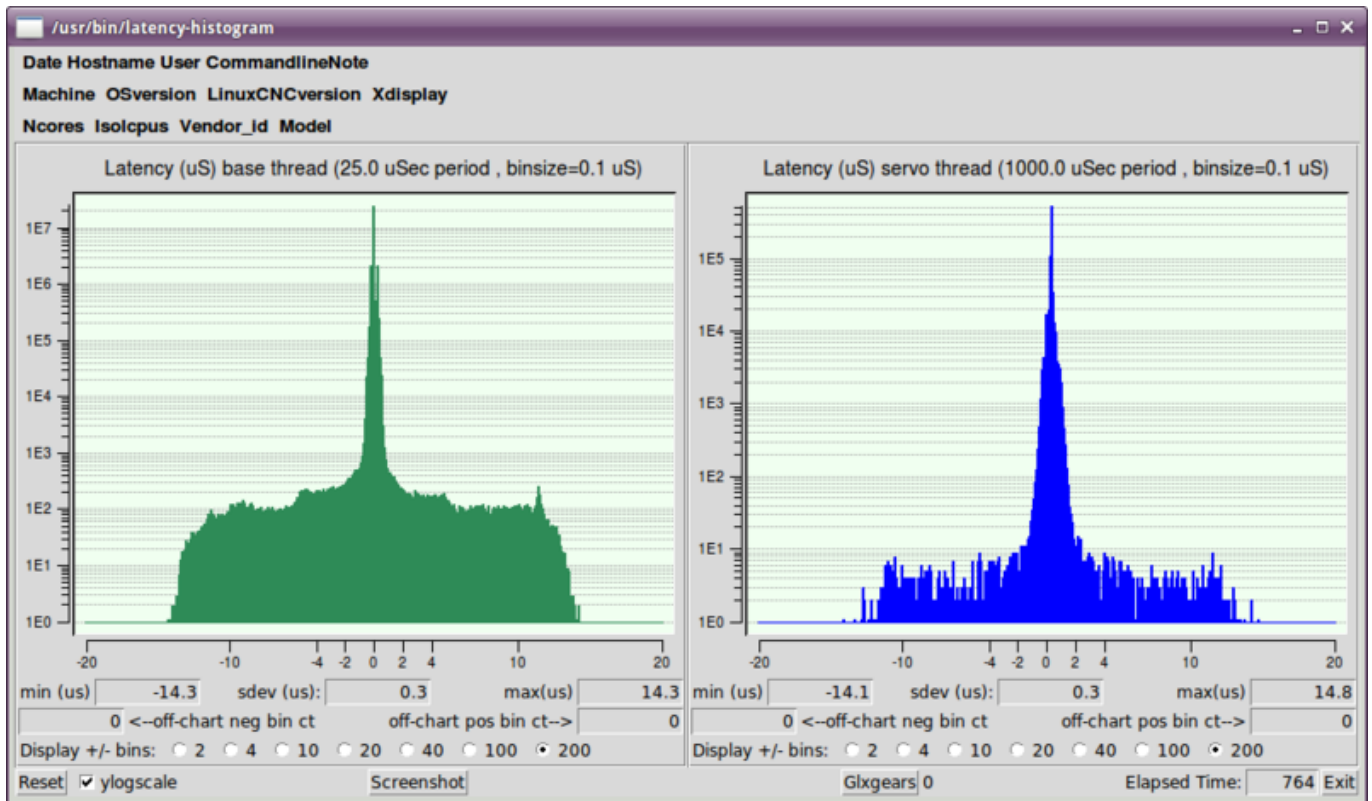


Figure 8.5: latency-histogram Window

8.3 Stepper Tuning

8.3.1 Getting the most out of Software Stepping

Generating step pulses in software has one very big advantage - it's free. Just about every PC has a parallel port that is capable of outputting step pulses that are generated by the software. However, software step pulses also have some disadvantages:

- limited maximum step rate
- jitter in the generated pulses
- loads the CPU

This chapter has some steps that can help you get the best results from software generated steps.

8.3.1.1 Run a Latency Test

The CPU is not the only factor determining latency. Motherboards, graphics cards, USB ports and many other things can degrade it. The best way to know what to expect from a PC is to run the RT latency tests.

Run the latency test as described in the [Latency Test](#) chapter.

While the test is running, you should *abuse* the computer. Move windows around on the screen. Surf the web. Copy some large files around on the disk. Play some music. Run an OpenGL program such

as glxgears. The idea is to put the PC through its paces while the latency test checks to see what the worst case numbers are.

The last number in the column labeled *Max Jitter* is the most important. Write it down - you will need it later. It contains the worst latency measurement during the entire run of the test. In the example above, that is 6693 nano-seconds, or 6,69 micro-seconds, which is excellent. However the example only ran for a few seconds (it prints one line every second). You should run the test for at least several minutes; sometimes the worst case latency doesn't happen very often, or only happens when you do some particular action. I had one Intel motherboard that worked pretty well most of the time, but every 64 seconds it had a very bad 300 μ s latency. Fortunately that is fixable, see [Fixing SMI issues on the LinuxCNC Wiki](#)

So, what do the results mean? If your *Max Jitter* number is less than about 15-20 microseconds (15000-20000 nanoseconds), the computer should give very nice results with software stepping. If the max latency is more like 30-50 microseconds, you can still get good results, but your maximum step rate might be a little disappointing, especially if you use microstepping or have very fine pitch leadscrews. If the numbers are 100 μ s or more (100,000 nanoseconds), then the PC is not a good candidate for software stepping. Numbers over 1 millisecond (1,000,000 nanoseconds) mean the PC is not a good candidate for LinuxCNC, regardless of whether you use software stepping or not.

Note that if you get high numbers, there may be ways to improve them. For example, one PC had very bad latency (several milliseconds) when using the onboard video. But a \$5 used video card solved the problem - LinuxCNC does not require bleeding edge hardware.

8.3.1.2 Figure out what your drives expect

Different brands of stepper drives have different timing requirements on their step and direction inputs. So you need to dig out (or Google for) the data sheet that has your drive's specs.

From the Gecko G202 manual:

```
Step Frequency: 0 to 200 kHz
Step Pulse "0" Time: 0.5  $\mu$ s min (Step on falling edge)
Step Pulse "1" Time: 4.5  $\mu$ s min
Direction Setup: 1  $\mu$ s min (20  $\mu$ s min hold time after Step edge)
```

From the Gecko G203V manual:

```
Step Frequency: 0 to 333 kHz
Step Pulse "0" Time: 2.0  $\mu$ s min (Step on rising edge)
Step Pulse "1" Time: 1.0  $\mu$ s min
```

Direction Setup:

```
200 ns (0.2  $\mu$ s) before step pulse rising edge
200 ns (0.2  $\mu$ s) hold after step pulse rising edge
```

From the Xylotex datasheet:

```
Minimum DIR setup time before rising edge of STEP Pulse 200 ns
Minimum DIR hold time after rising edge of STEP pulse 200 ns
Minimum STEP pulse high time 2.0  $\mu$ s
Minimum STEP pulse low time 1.0  $\mu$ s
Step happens on rising edge
```

Once you find the numbers, write them down too - you need them in the next step.

8.3.1.3 Choose your BASE_PERIOD

BASE_PERIOD is the *heartbeat* of your LinuxCNC computer. Every period, the software step generator decides if it is time for another step pulse. A shorter period will allow you to generate more pulses per second, within limits. But if you go too short, your computer will spend so much time generating step pulses that everything else will slow to a crawl, or maybe even lock up. Latency and stepper drive requirements affect the shortest period you can use, as we will see in a minute.

Let's look at the Gecko example first. The G202 can handle step pulses that go low for 0.5 μ s and high for 4.5 μ s, it needs the direction pin to be stable 1 μ s before the falling edge, and remain stable for 20 μ s after the falling edge. The longest timing requirement is the 20 μ s hold time. A simple approach would be to set the period at 20 μ s. That means that all changes on the STEP and DIR lines are separated by 20 μ s. All is good, right?

Wrong! If there was ZERO latency, then all edges would be separated by 20 μ s, and everything would be fine. But all computers have some latency. Latency means lateness. If the computer has 11 μ s of latency, that means sometimes the software runs as much as 11 μ s later than it was supposed to. If one run of the software is 11 μ s late, and the next one is on time, the delay from the first to the second is only 9 μ s. If the first one generated a step pulse, and the second one changed the direction bit, you just violated the 20 μ s G202 hold time requirement. That means your drive might have taken a step in the wrong direction, and your part will be the wrong size.

The really nasty part about this problem is that it can be very very rare. Worst case latencies might only happen a few times a minute, and the odds of bad latency happening just as the motor is changing direction are low. So you get very rare errors that ruin a part every once in a while and are impossible to troubleshoot.

The simplest way to avoid this problem is to choose a BASE_PERIOD that is the sum of the longest timing requirement of your drive, and the worst case latency of your computer. If you are running a Gecko with a 20 μ s hold time requirement, and your latency test said you have a maximum latency of 11 μ s, then if you set the BASE_PERIOD to $20+11 = 31$ μ s (31000 nano-seconds in the ini file), you are guaranteed to meet the drive's timing requirements.

But there is a tradeoff. Making a step pulse requires at least two periods. One to start the pulse, and one to end it. Since the period is 31 μ s, it takes $2 \times 31 = 62$ μ s to create a step pulse. That means the maximum step rate is only 16,129 steps per second. Not so good. (But don't give up yet, we still have some tweaking to do in the next section.)

For the Xylotex, the setup and hold times are very short, 200 ns each (0.2 μ s). The longest time is the 2 μ s high time. If you have 11 μ s latency, then you can set the BASE_PERIOD as low as $11+2=13$ μ s. Getting rid of the long 20 μ s hold time really helps! With a period of 13 μ s, a complete step takes $2 \times 13 = 26$ μ s, and the maximum step rate is 38,461 steps per second!

But you can't start celebrating yet. Note that 13 μ s is a very short period. If you try to run the step generator every 13 μ s, there might not be enough time left to run anything else, and your computer will lock up. If you are aiming for periods of less than 25 μ s, you should start at 25 μ s or more, run LinuxCNC, and see how things respond. If all is well, you can gradually decrease the period. If the mouse pointer starts getting sluggish, and everything else on the PC slows down, your period is a little too short. Go back to the previous value that let the computer run smoothly.

In this case, suppose you started at 25 μ s, trying to get to 13 μ s, but you find that around 16 μ s is the limit - any less and the computer doesn't respond very well. So you use 16 μ s. With a 16 μ s period and 11 μ s latency, the shortest output time will be $16-11 = 5$ μ s. The drive only needs 2 μ s, so you have some margin. Margin is good - you don't want to lose steps because you cut the timing too close.

What is the maximum step rate? Remember, two periods to make a step. You settled on 16 μ s for the period, so a step takes 32 μ s. That works out to a not bad 31,250 steps per second.

8.3.1.4 Use steplen, stepspace, dirsetup, and/or dirhold

In the last section, we got the Xylotex drive to a 16 μ s period and a 31,250 step per second maximum speed. But the Gecko was stuck at 31 μ s and a not-so-nice 16,129 steps per second. The Xylotex

example is as good as we can make it. But the Gecko can be improved.

The problem with the G202 is the 20 μs hold time requirement. That plus the 11 μs latency is what forces us to use a slow 31 μs period. But the LinuxCNC software step generator has some parameters that let you increase the various time from one period to several. For example, if `steplen` is changed from 1 to 2, then there will be two periods between the beginning and end of the step pulse. Likewise, if `dirhold` is changed from 1 to 3, there will be at least three periods between the step pulse and a change of the direction pin.

If we can use `dirhold` to meet the 20 μs hold time requirement, then the next longest time is the 4.5 μs high time. Add the 11 μs latency to the 4.5 μs high time, and you get a minimum period of 15.5 μs . When you try 15.5 μs , you find that the computer is sluggish, so you settle on 16 μs . If we leave `dirhold` at 1 (the default), then the minimum time between step and direction is the 16 μs period minus the 11 μs latency = 5 μs , which is not enough. We need another 15 μs . Since the period is 16 μs , we need one more period. So we change `dirhold` from 1 to 2. Now the minimum time from the end of the step pulse to the changing direction pin is $5 + 16 = 21$ μs , and we don't have to worry about the Gecko stepping the wrong direction because of latency.

If the computer has a latency of 11 μs , then a combination of a 16 μs base period, and a `dirhold` value of 2 ensures that we will always meet the timing requirements of the Gecko. For normal stepping (no direction change), the increased `dirhold` value has no effect. It takes two periods totalling 32 μs to make each step, and we have the same 31,250 step per second rate that we got with the Xylotex.

The 11 μs latency number used in this example is very good. If you work through these examples with larger latency, like 20 or 25 μs , the top step rate for both the Xylotex and the Gecko will be lower. But the same formulas apply for calculating the optimum `BASE_PERIOD`, and for tweaking `dirhold` or other step generator parameters.

8.3.1.5 No Guessing!

For a fast AND reliable software based stepper system, you cannot just guess at periods and other configuration parameters. You need to make measurements on your computer, and do the math to ensure that your drives get the signals they need.

To make the math easier, I've created an Open Office spreadsheet [Step Timing Calculator](#). You enter your latency test result and your stepper drive timing requirements and the spreadsheet calculates the optimum `BASE_PERIOD`. Next, you test the period to make sure it won't slow down or lock up your PC. Finally, you enter the actual period, and the spreadsheet will tell you the stepgen parameter settings that are needed to meet your drive's timing requirements. It also calculates the maximum step rate that you will be able to generate.

I've added a few things to the spreadsheet to calculate max speed and stepper electrical calculations.

8.4 Stepper Diagnostics

If what you get is not what you expect many times you just got some experience. Learning from the experience increases your understanding of the whole. Diagnosing problems is best done by divide and conquer. By this I mean if you can remove 1/2 of the variables from the equation each time you will find the problem the fastest. In the real world this is not always the case, but it's usually a good place to start.

8.4.1 Common Problems

8.4.1.1 Stepper Moves One Step

The most common reason in a new installation for a stepper motor not to move is that the step and direction signals are exchanged. If you press the jog forward and jog backward keys, alternately, and

the stepper moves one step each time, and in the same direction, there is your clue.

8.4.1.2 No Steppers Move

Many drives have an enable pin or need a charge pump to enable the output.

8.4.1.3 Distance Not Correct

If you command the axis to move a specific distance and it does not move that distance, then your scale setting is wrong.

8.4.2 Error Messages

8.4.2.1 Following Error

The concept of a following error is strange when talking about stepper motors. Since they are an open loop system, there is no position feedback to let you know if you actually are out of range. LinuxCNC calculates if it can keep up with the motion called for, and if not, then it gives a following error. Following errors usually are the result of one of the following on stepper systems.

- FERROR too small
- MIN_FERROR too small
- MAX_VELOCITY too fast
- MAX_ACCELERATION too fast
- BASE_PERIOD set too long
- Backlash added to an axis

Any of the above can cause the real-time pulsing to not be able to keep up the requested step rate. This can happen if you didn't run the latency test long enough to get a good number to plug into the Stepconf Wizard, or if you set the Maximum Velocity or Maximum Acceleration too high.

If you added backlash you need to increase the STEPGEN_MAXACCEL up to double the MAX_ACCELERATION in the AXIS section of the INI file for each axis you added backlash to. LinuxCNC uses "extra acceleration" at a reversal to take up the backlash. Without backlash correction, step generator acceleration can be just a few percent above the motion planner acceleration.

8.4.2.2 RTAPI Error

When you get this error:

```
RTAPI: ERROR: Unexpected realtime delay on task n
```

This error is generated by rtapi based on an indication from RTAI that a deadline was missed. It is usually an indication that the BASE_PERIOD in the [EMCMOT] section of the ini file is set too low. You should run the Latency Test for an extended period of time to see if you have any delays that would cause this problem. If you used the Stepconf Wizard, run it again, and test the Base Period Jitter again, and adjust the Base Period Maximum Jitter on the Basic Machine Information page. You might have to leave the test running for an extended period of time to find out if some hardware causes intermittent problems.

LinuxCNC tracks the number of CPU cycles between invocations of the real-time thread. If some element of your hardware is causing delays or your realtime threads are set too fast you will get this error.

Note

This error is only displayed once per session. If you had your BASE_PERIOD too low you could get hundreds of thousands of error messages per second if more than one was displayed.

8.4.3 Testing

8.4.3.1 Step Timing

If you are seeing an axis ending up in the wrong location over multiple moves, it is likely that you do not have the correct direction hold times or step timing for your stepper drivers. Each direction change may be losing a step or more. If the motors are stalling, it is also possible you have either the MAX_ACCELERATION or MAX_VELOCITY set too high for that axis.

The following program will test the Z axis configuration for proper setup. Copy the program to your ~/emc2/nc_files directory and name it TestZ.ngc or similar. Zero your machine with Z = 0.000 at the table top. Load and run the program. It will make 200 moves back and forth from 0.5 to 1". If you have a configuration issue, you will find that the final position will not end up 0.500" that the axis window is showing. To test another axis just replace the Z with your axis in the G0 lines.

```
( test program to see if Z axis loses position )
( msg, test 1 of Z axis configuration )
G20 #1000=100 ( loop 100 times )
( this loop has delays after moves )
( tests acc and velocity settings )
o100 while [#1000]
    G0 Z1.000
    G4 P0.250
    G0 Z0.500
    G4 P0.250
    #1000 = [#1000 - 1]
o100 endwhile
( msg, test 2 of Z axis configuration S to continue)
M1 (stop here)
#1000=100 ( loop 100 times )
( the next loop has no delays after moves )
( tests direction hold times on driver config and also max accel setting )
o101 while [#1000]
    G0 Z1.000
    G0 Z0.500
    #1000 = [#1000 - 1]
o101 endwhile
( msg, Done...Z should be exactly .5" above table )
M2
```


Chapter 9

Configuration

9.1 Stepper Quickstart

This section assumes you have done a standard install from the Live CD. After installation it is recommended that you connect the computer to the Internet and wait for the update manager to pop up and get the latest updates for LinuxCNC and Ubuntu before continuing.

9.1.1 Latency Test

The Latency Test determines how late your computer processor is in responding to a request. Some hardware can interrupt the processing which could cause missed steps when running a CNC machine. This is the first thing you need to do. Follow the instructions [here](#) to run the latency test.

9.1.2 Sherline

If you have a Sherline several predefined configurations are provided. This is on the main menu CNC/EMC then pick the Sherline configuration that matches yours and save a copy.

9.1.3 Xylotex

If you have a Xylotex you can skip the following sections and go straight to the [Stepper Config Wizard](#). LinuxCNC has provided quick setup for the Xylotex machines.

9.1.4 Machine Information

Gather the information about each axis of your machine.

Drive timing is in nano seconds. If you're unsure about the timing many popular drives are included in the stepper configuration wizard. Note some newer Gecko drives have different timing than the original one. A [list](#) is also on the user maintained LinuxCNC wiki site of more drives.

Axis	Drive Type	Step Time ns	Step Space ns	Dir. Hold ns	Dir. Setup ns
X					
Y					

Axis	Drive Type	Step Time ns	Step Space ns	Dir. Hold ns	Dir. Setup ns
Z					

9.1.5 Pinout Information

Gather the information about the connections from your machine to the PC parallel port.

Output Pin	Typ. Function	If Different	Input Pin	Typ. Function	If Different
1	E-Stop Out		10	X Limit/Home	
2	X Step		11	Y Limit/Home	
3	X Direction		12	Z Limit/Home	
4	Y Step		13	A Limit/Home	
5	Y Direction		15	Probe In	
6	Z Step				
7	Z Direction				
8	A Step				
9	A Direction				
14	Spindle CW				
16	Spindle PWM				
17	Amplifier Enable				

Note any pins not used should be set to Unused in the drop down box. These can always be changed later by running Stepconf again.

9.1.6 Mechanical Information

Gather information on steps and gearing. The result of this is steps per user unit which is used for SCALE in the .ini file.

Axis	Steps/Rev.	Micro Steps	Motor Teeth	Leadscrew Teeth	Leadscrew Pitch
X					
Y					
Z					

- *Steps per revolution* - is how many stepper-motor-steps it takes to turn the stepper motor one revolution. Typical is 200.
- *Micro Steps* - is how many steps the drive needs to move the stepper motor one full step. If microstepping is not used, this number will be 1. If microstepping is used the value will depend on the stepper drive hardware.
- *Motor Teeth and Leadscrew Teeth* - is if you have some reduction (gears, chain, timing belt, etc.) between the motor and the leadscrew. If not, then set these both to 1.
- *Leadscrew Pitch* - is how much movement occurs (in user units) in one leadscrew turn. If you're setting up in inches then it is inches per turn. If you're setting up in millimeters then it is millimeters per turn.

The net result you're looking for is how many CNC-output-steps it takes to move one user unit (inches or mm).

Example 9.1 Units inches

Stepper = 200 steps per revolution
 Drive = 10 micro steps per step
 Motor Teeth = 20
 Leadscrew Teeth = 40
 Leadscrew Pitch = 0.2000 inches per turn

From the above information, the leadscrew moves 0.200 inches per turn. - The motor turns 2.000 times per 1 leadscrew turn. - The drive takes 10 microstep inputs to make the stepper step once. - The drive needs 2000 steps to turn the stepper one revolution.

So the scale needed is:

$$\frac{200\text{motor steps}}{1\text{motor rev}} \times \frac{10\text{microsteps}}{1\text{motor step}} \times \frac{2\text{motor revs}}{1\text{leadscrew rev}} \times \frac{1\text{leadscrew rev}}{0.2000\text{inch}} = \frac{20,000\text{microsteps}}{\text{inch}}$$

Example 9.2 Units mm

Stepper = 200 steps per revolution
 Drive = 8 micro steps per step
 Motor Teeth = 30
 Leadscrew Teeth = 90
 Leadscrew Pitch = 5.00 mm per turn

From the above information: - The leadscrew moves 5.00 mm per turn. - The motor turns 3.000 times per 1 leadscrew turn. - The drive takes 8 microstep inputs to make the stepper step once. - The drive needs 1600 steps to turn the stepper one revolution.

So the scale needed is:

$$\frac{200\text{full steps}}{1\text{rev}} \times \frac{8\text{microsteps}}{1\text{step}} \times \frac{3\text{revs}}{1\text{leadscrew rev}} \times \frac{1\text{leadscrew rev}}{5.00\text{mm}} = \frac{960\text{steps}}{1\text{mm}}$$

9.2 INI Configuration

9.2.1 The INI File Components

A typical INI file follows a rather simple layout that includes;

- comments
- sections
- variables

Each of these elements is separated on single lines. Each end of line or newline character creates a new element.

9.2.1.1 Comments

A comment line is started with a `;` or a `#` mark. When the INI reader sees either of these marks at the start a line, the rest of the line is ignored by the software. Comments can be used to describe what an INI element will do.

```
; This is my mill configuration file.
# I set it up on January 12, 2012
```

Comments can also be used to *turn off* a variable. This makes it easier to pick between different variables.

```
DISPLAY = axis
# DISPLAY = touchy
```

In this list, the DISPLAY variable will be set to axis because the other one is commented out. If someone carelessly edits a list like this and leaves two of the lines uncommented, the first one encountered will be used.

Note that inside a variable, the `"#"` and `;"` characters do not denote comments:

```
INCORRECT = value      # and a comment

# Correct Comment
CORRECT = value
```

9.2.1.2 Sections

Related parts of an INI file are separated into sections. A section name is enclosed in brackets like this `[THIS_SECTION]`. The order of sections is unimportant. Sections begin at the section name and end at the next section name.

The following sections are used by LinuxCNC:

- `<<sub:ini:sec:emc,[EMC]>>` general information
- `<<sub:ini:sec:display,[DISPLAY]>>` settings related to the graphical user interface
- `<<sub:ini:sec:filter,[FILTER]>>` settings input filter programs
- `<<sub:ini:sec:rs274ngc,[RS274NGC]>>` settings used by the G-code interpreter
- `<<sub:ini:sec:emcmot,[EMCMOT]>>` settings used by the real time motion controller
- `<<sub:ini:sec:task,[TASK]>>` settings used by the task controller
- `<<sub:ini:sec:hal,[HAL]>>` specifies .hal files
- `<<sub:ini:sec:halui,[HALUI]>>` MDI commands used by HALUI
- `<<sub:ini:sec:applications,[APPLICATIONS]>>` Other applications to be started by LinuxCNC
- `<<sub:ini:sec:traj,[TRAJ]>>` additional settings used by the real time motion controller
- `<<sub:ini:sec:joint-num,[JOINT_n]>>` individual joint variables
- `<<sub:ini:sec:axis-letter,[AXIS_l]>>` individual axis variables
- `<<sub:ini:sec:kins,[KINS]>>` kinematics variables
- `<<sub:ini:sec:emcio,[EMCIO]>>` settings used by the I/O Controller

9.2.1.3 Variables

A variable line is made up of a variable name, an equals sign (=), and a value. Everything from the first non-white space character after the = up to the end of the line is passed as the value, so you can embed spaces in string symbols if you want to or need to. A variable name is often called a keyword.

Variable Example

```
MACHINE = My Machine
```

A variable line may be extended to multiple lines with a terminal backslash (\) character. A maximum of MAX_EXTEND_LINES (==20) are allowed. There must be no whitespace following the trailing backslash character.

Section identifiers may not be extended to multiple lines.

Variable with Line extends Example

```
APP = sim_pin \
ini.0.max_acceleration \
ini.1.max_acceleration \
ini.2.max_acceleration \
ini.0.max_velocity \
ini.1.max_velocity \
ini.2.max_velocity
```

The following sections detail each section of the configuration file, using sample values for the configuration lines.

Variables that are used by LinuxCNC must always use the section names and variable names as shown. In the following example the variable *MACHINE* is assigned the value *My Machine*.

9.2.1.4 Custom Sections and Variables

Most sample configurations use custom sections and variables to put all of the settings into one location for convenience.

To add a custom variable to an existing LinuxCNC section, simply include the variable in that section.

Custom Variable Example

```
[JOINT_0]
TYPE = LINEAR
...
SCALE = 16000
```

To introduce a custom section with its own variables, add the section and variables to the INI file.

Custom Section Example

```
[PROBE]
Z_FEEDRATE = 50
Z_OFFSET = 12
Z_SAFE_DISTANCE = -10
```

To use the custom variables in your HAL file, put the section and variable name in place of the value.

HAL Example

```
setp offset.1.offset [PROBE]Z_OFFSET
setp stepgen.0.position-scale [JOINT_0]SCALE
```

Note

The value stored in the variable must match the type specified by the component pin.

To use the custom variables in G-code, use the global variable syntax `#<_ini[section]variable>`. The following example shows a simple Z-axis touch-off routine for a router or mill using a probe plate.

G-code Example

```
G91
G38.2 Z#<_ini[probe]z_safe_distance> F#<_ini[probe]z_feedrate>
G90
G1 Z#5063
G10 L20 P0 Z#<_ini[probe]z_offset>
```

9.2.1.5 Include Files

An INI file may include the contents of another file by using a `#INCLUDE` directive.

#INCLUDE Format

```
#INCLUDE filename
```

The filename can be specified as:

- a file in the same directory as the INI file
- a file located relative to the working directory
- an absolute file name (starts with a /)
- a user-home-relative file name (starts with a ~)

Multiple `#INCLUDE` directives are supported.

#INCLUDE Examples

```
#INCLUDE joint_0.inc
#INCLUDE ../parallel/joint_1.inc
#INCLUDE below/joint_2.inc
#INCLUDE /home/myusername/myincludes/display.inc
#INCLUDE ~/linuxcnc/myincludes/rs274ngc.inc
```

The `#INCLUDE` directives are supported for one level of expansion only — an included file may not include additional files. The recommended file extension is `.inc`. Do not use a file extension of `.ini` for included files.

9.2.2 INI File Sections

9.2.2.1 [EMC] Section

- `VERSION = 1.1` - The version number for the configuration. Any value other than 1.1 will cause the configuration checker to run and try to update the configuration to the new style joint axes type of configuration.
 - `MACHINE = My Controller` - This is the name of the controller, which is printed out at the top of most graphical interfaces. You can put whatever you want here as long as you make it a single line long.
 - `DEBUG = 0` - Debug level 0 means no messages will be printed when LinuxCNC is run from a [terminal](#). Debug flags are usually only useful to developers. See `src/emc/nml_intf/debugflags.h` for other settings.
-

9.2.2.2 [DISPLAY] Section

Different user interface programs use different options, and not every option is supported by every user interface. There are several interfaces, like Axis, Gmoccapy, Touchy, QtVCP's QtDragon and Gscreen. Axis is an interface for use with normal computer and monitor, Touchy is for use with touch screens. Gmoccapy can be used both ways and offers also many connections for hardware controls. Descriptions of the interfaces are in the Interfaces section of the User Manual.

- *DISPLAY = axis* - The name of the user interface to use. Valid options may include: *axis*, *touchy*, *gmoccapy*, *gscreen*, *tklinuxcnc*, *qtvcp*
- *POSITION_OFFSET = RELATIVE* - The coordinate system (RELATIVE or MACHINE) to show on the DRO when the user interface starts. The RELATIVE coordinate system reflects the G92 and G5x coordinate offsets currently in effect.
- *POSITION_FEEDBACK = COMMANDED* - The coordinate value (COMMANDED or ACTUAL) to show on the DRO when the user interface starts. In Axis this can be changed from the View menu. The COMMANDED position is the position requested by LinuxCNC. The ACTUAL position is the feedback position of the motors if they have feedback like most servo systems. Typically the COMMANDED value is used.
- *DRO_FORMAT_MM = %+08.6f* - Override the default DRO formatting in metric mode (normally 3 decimal places, padded with spaces to 6 digits to the left). The example above will pad with zeros, display 6 decimal digits and force display of a + sign for positive numbers. Formatting follows Python practice. <https://docs.python.org/2/library/string.html#format-specification-mini-language> an error will be raised if the format can not accept a floating-point value.
- *DRO_FORMAT_IN = % 4.1f* - Override the default DRO formatting in imperial mode (normally 4 decimal places, padded with spaces to 6 digits to the left). The example above will display only one decimal digit. Formatting follows Python practice. <https://docs.python.org/2/library/string.html#format-specification-mini-language> An error will be raised if the format can not accept a floating-point value.
- *CONE_BASESIZE = .25* - Override the default cone/tool base size of .5 in the graphics display
- *MAX_FEED_OVERRIDE = 1.2* - The maximum feed override the user may select. 1.2 means 120% of the programmed feed rate.
- *MIN_SPINDLE_OVERRIDE = 0.5* - The minimum spindle override the user may select. 0.5 means 50% of the programmed spindle speed. (This is used to set the minimum spindle speed).
- *MIN_SPINDLE_0_OVERRIDE = 0.5* - The minimum spindle override the user may select. 0.5 means 50% of the programmed spindle speed. (This is used to set the minimum spindle speed). On multi spindle machine there will be entries for each spindle number. Only used by the QtVCP based user interfaces.
- *MAX_SPINDLE_OVERRIDE = 1.0* - The maximum spindle override the user may select. 1.0 means 100% of the programmed spindle speed.
- *MAX_SPINDLE_0_OVERRIDE = 1.0* - The maximum feed override the user may select. 1.2 means 120% of the programmed feed rate. On multi spindle machine there will be entries for each spindle number. Only used by the QtVCP based user interfaces.
- *DEFAULT_SPINDLE_SPEED = 100* - The default spindle RPM when the spindle is started in manual mode. if this setting is not present, this defaults to 1 RPM for AXIS and 300 RPM for gmoccapy.
- *deprecated* - use the [SPINDLE_n] section instead
- *DEFAULT_SPINDLE_0_SPEED = 100* - The default spindle RPM when the spindle is started in manual mode. On multi spindle machine there will be entries for each spindle number. Only used by the QtVCP based user interfaces.

- *deprecated* - use the [SPINDLE_n] section instead.
- *SPINDLE_INCREMENT = 200* - The increment used when clicking increase/decrease buttons Only used by the QtVCP based user interfaces.
- *deprecated* - use the [SPINDLE_n] section instead.
- *MIN_SPINDLE_0_SPEED = 1000* - The minimum RPM that can be manually selected. On multi spindle machine there will be entries for each spindle number. Only used by the QtVCP based user interfaces.
- *deprecated* - use the [SPINDLE_n] section instead.
- *MAX_SPINDLE_0_SPEED = 20000* - The maximum RPM that can be manually selected. On multi spindle machine there will be entries for each spindle number. Only used by the QtVCP based user interfaces.
- *deprecated* - use the [SPINDLE_n] section instead.
- *PROGRAM_PREFIX = ~/linuxcnc/nc_files* - The default location for G-code files and the location for user-defined M-codes. This location is searched for the file name before the subroutine path and user M path if specified in the [RS274NGC] section.
- *INTRO_GRAPHIC = emc2.gif* - The image shown on the splash screen.
- *INTRO_TIME = 5* - The maximum time to show the splash screen, in seconds.
- *CYCLE_TIME = 100* - Cycle time of the Display GUI. Depending on the screen, this can be in seconds or ms (ms preferred). This is often the update rate rather than sleep time between updates. If the update time is not set right the screen can become unresponsive or very jerky. A value of 100ms (0.1 seconds) is a common setting though a range of 50 - 200ms (.05 - .2 seconds) may be useable. An under powered CPU may see improvement with a longer setting. Usually the default is fine.
- *PREVIEW_TIMEOUT = 5* - Timeout (in seconds) for loading graphical preview of G-code. Currently AXIS only.

Note

The following [DISPLAY] items are used by GladeVCP, see the [embedding a tab](#) section of the GladeVCP Chapter.

- *EMBED_TAB_NAME=GladeVCP demo*
-

Note

Different user interface programs use different options, and not every option is supported by every user interface. See [AXIS GUI](#) document for AXIS details. See [gmoccapy](#) document for Gmoccapy details.

- *DEFAULT_LINEAR_VELOCITY = .25* - The default velocity for linear jogs, in [machine units](#) per second.
 - *MIN_VELOCITY = .01* - The approximate lowest value the jog slider.
 - *MAX_LINEAR_VELOCITY = 1.0* - The maximum velocity for linear jogs, in machine units per second.
 - *MIN_LINEAR_VELOCITY = .01* - The approximate lowest value the jog slider.
-

- *DEFAULT_ANGULAR_VELOCITY* = .25 - The default velocity for angular jogs, in machine units per second.
- *MIN_ANGULAR_VELOCITY* = .01 - The approximate lowest value the angular jog slider.
- *MAX_ANGULAR_VELOCITY* = 1.0 - The maximum velocity for angular jogs, in machine units per second.
- *INCREMENTS* = 1 mm, .5 in, ... - Defines the increments available for incremental jogs. The INCREMENTS can be used to override the default. The values can be decimal numbers (e.g., 0.1000) or fractional numbers (e.g., 1/16), optionally followed by a unit (cm, mm, um, inch, in or mil). If a unit is not specified the machine unit is assumed. Metric and imperial distances may be mixed: INCREMENTS = 1 inch, 1 mil, 1 cm, 1 mm, 1 um is a valid entry.
- *GRIDS* = 10 mm, 1 in, ... - Defines the preset values for grid lines. The value is interpreted the same way as *INCREMENTS*.
- *OPEN_FILE* = /full/path/to/file.ngc - The file to show in the preview plot when AXIS starts. Use a blank string "" and no file will be loaded at start up. gmoccapy will not use this setting, as it offers a corresponding entry on its settings page.
- *EDITOR* = gedit - The editor to use when selecting File > Edit to edit the G-code from the AXIS menu. This must be configured for this menu item to work. Another valid entry is gnome-terminal -e vim. This entry does not apply to gmoccapy, as gmoccapy has an integrated editor.
- *TOOL_EDITOR* = tooledit - The editor to use when editing the tool table (for example by selecting "File > Edit tool table..." in Axis). Other valid entries are "gedit", "gnome-terminal -e vim", and "gvim". This entry does not apply to gmoccapy, as gmoccapy has an integrated editor.
- *PYVCP* = /filename.xml - The PyVCP panel description file. See the [PyVCP Chapter](#) for more information.
- *PYVCP_POSITION* = BOTTOM - The placement of the PyVCP panel in the AXIS user interface. If this variable is omitted the panel will default to the right side. The only valid alternative is BOTTOM. See the [PyVCP Chapter](#) for more information.
- *LATHE* = 1 - Any non-empty value (including "0") causes axis to use "lathe mode" with a top view and with Radius and Diameter on the DRO.
- *BACK_TOOL_LATHE* = 1 - Any non-empty value (including "0") causes axis to use "back tool lathe mode" with inverted X axis.
- *FOAM* = 1 - Any non-empty value (including "0") causes axis to change the display for foam-cutter mode.
- *GEOMETRY* = XYZABCUVW - Controls the **preview** and **backplot** of motion. This item consists of a sequence of axis letters and control characters, optionally preceded with a "-" sign:
 1. The letters X, Y, Z specify translation along the named coordinate.
 2. The letters A, B, C specify rotation about the corresponding axes X, Y, Z.
 3. The letters U, V, W specify translation along the related axes X, Y, Z.
 4. Each letter specified must occur in [TRAJ]COORDINATES to have an effect.
 5. A "-" character preceding any letter inverts the direction of the operation.
 6. The translation and rotation operations are evaluated **right-to-left**. So using GEOMETRY=XYZBC specifies a C rotation followed by a B rotation followed by Z, Y, X translations. (The ordering of consecutive translation letters is immaterial.)
 7. If the "!" special character appears anywhere in the sequence, rotations for the A, B, C axis letters respect the offsets (G5x, G92) applied to X, Y, Z.

8. The proper GEOMETRY string depends on the machine configuration and the kinematics used to control it. The order of the letters is important. For example, rotating around C then B is different than rotating around B then C.
9. Rotations are by default applied with respect to the machine origin. Example: GEOMETRY=CXYZ first translates the control point to X, Y, Z and then performs a C rotation about the Z axis centered at the machine origin.
10. Rotations applied after translations may use the "!" provision to act with respect to the current machine offset. Example: GEOMETRY=!CXYZ translates the control point to the X, Y, Z position and then performs a C rotation about the machine origin displaced by the current G5x,G92 X, Y, Z offsets. (Changing offsets may require a program reload).
11. UVW translation example: GEOMETRY=XYZUVW causes UVW to move in the coordinate system of the tool and XYZ to move in the coordinate system of the material.
12. Foam-cutting machines (FOAM = 1) should specify "XY;UV" or leave the value blank even though this value is presently ignored in foam-cutter mode. A future version may define what ";" means, but if it does "XY;UV" will mean the same as the current foam default.

Note

If no [DISPLAY]GEOMETRY is included in the INI file, a default is provided by the [DISPLAY]DISPLAY GUI program (typically "XYZABCUVW")

- **ARCDIVISION = 64** - Set the quality of preview of arcs. Arcs are previewed by dividing them into a number of straight lines; a semicircle is divided into **ARCDIVISION** parts. Larger values give a more accurate preview, but take longer to load and result in a more sluggish display. Smaller values give a less accurate preview, but take less time to load and may result in a faster display. The default value of 64 means a circle of up to 3 inches will be displayed to within 1 mil (.03%).
- **MDI_HISTORY_FILE =** - The name of a local MDI history file. If this is not specified Axis will save the MDI history in **.axis_mdi_history** in the user's home directory. This is useful if you have multiple configurations on one computer.
- **JOG_AXES =** - The order in which jog keys are assigned to axis letters. The left and right arrows are assigned to the first axis letter, up and down to the second, page up/page down to the third, and left and right bracket to the fourth. If unspecified, the default is determined from the [TRAJ]COORDINATES, [DISPLAY]LATHE and [DISPLAY]FOAM values.
- **JOG_INVERT =** - For each axis letter, the jog direction is inverted. The default is "X" for lathes and blank otherwise.

Note

The settings for **JOG_AXES** and **JOG_INVERT** apply to world mode jogging by axis coordinate letter and are in effect while in world mode after successful homing. When operating in joint mode prior to homing, keyboard jog keys are assigned in a fixed sequence: left/right: joint0, up/down: joint1, page up/page down: joint2, left/right bracket: joint3

- **USER_COMMAND_FILE = mycommands.py** - The name of an optional, configuration-specific python file sourced by the axis GUI instead of the user-specific file ~/.axisrc.

Note

The following [DISPLAY] item is used by the TKLinuxCNC interface only.

- **HELP_FILE = tklinucnc.txt** - Path to help file.
-

9.2.2.3 [FILTER] Section

AXIS and gmoccapy have the ability to send loaded files through a filter program. This filter can do any desired task: Something as simple as making sure the file ends with M2, or something as complicated as detecting whether the input is a depth image, and generating G-code to mill the shape it defines. The [FILTER] section of the INI file controls how filters work. First, for each type of file, write a PROGRAM_EXTENSION line. Then, specify the program to execute for each type of file. This program is given the name of the input file as its first argument, and must write RS274NGC code to standard output. This output is what will be displayed in the text area, previewed in the display area, and executed by LinuxCNC when Run.

- PROGRAM_EXTENSION = .extension Description

If your post processor outputs files in all caps you might want to add the following line:

```
PROGRAM_EXTENSION = .NGC XYZ Post Processor
```

The following lines add support for the image-to-G-code converter included with LinuxCNC.

```
PROGRAM_EXTENSION = .png,.gif,.jpg # Greyscale Depth Image
png = image-to-gcode
gif = image-to-gcode
jpg = image-to-gcode
```

An example of a custom G-code converter located in the linuxcnc directory.

```
PROGRAM_EXTENSION = .gcode 3D Printer
gcode = /home/mill/linuxcnc/convert.py
```

Note

The program file associated with an extension must have either the full path to the program or be located in a directory that is on the system path.

It is also possible to specify an interpreter:

```
PROGRAM_EXTENSION = .py Python Script
py = python
```

In this way, any Python script can be opened, and its output is treated as G-code. One such example script is available at `nc_files/holecircle.py`. This script creates G-code for drilling a series of holes along the circumference of a circle. Many more G-code generators are on the LinuxCNC Wiki site <http://wiki.linuxcnc.org/>.

Python filters should use the print function to output the result to Axis.

This example program filters a file and adds a W axis to match the Z axis. It depends on there being a space between each axis word to work.

```
#!/usr/bin/env python3

import sys

def main(argv):

    openfile = open(argv[0], 'r')
    file_in = openfile.readlines()
    openfile.close()
```

```

file_out = []
for line in file_in:
    # print(line)
    if line.find('Z') != -1:
        words = line.rstrip('\n')
        words = words.split(' ')
        newword = ''
        for i in words:
            if i[0] == 'Z':
                newword = 'W'+ i[1:]
            if len(newword) > 0:
                words.append(newword)
                newline = ' '.join(words)
                file_out.append(newline)
        else:
            file_out.append(line)
for item in file_out:
    print("%s" % item)

if __name__ == "__main__":
    main(sys.argv[1:])

```

- **FILTER_PROGRESS=%d**

If the environment variable `AXIS_PROGRESS_BAR` is set, then lines written to stderr of the form above sets the `AXIS` progress bar to the given percentage. This feature should be used by any filter that runs for a long time.

9.2.2.4 [RS274NGC] Section

- `PARAMETER_FILE = myfile.var` - The file located in the same directory as the INI file which contains the parameters used by the interpreter (saved between runs).
- `ORIENT_OFFSET = 0` - A float value added to the R word parameter of an [M19 Orient Spindle](#) operation. Used to define an arbitrary zero position regardless of encoder mount orientation.
- `RS274NGC_STARTUP_CODE = G17 G20 G40 G49 G64 P0.001 G80 G90 G92 G94 G97 G98` - A string of NC codes that the interpreter is initialized with. This is not a substitute for specifying modal G-codes at the top of each ngc file, because the modal codes of machines differ, and may be changed by G-code interpreted earlier in the session.
- `SUBROUTINE_PATH = ncsubroutines:/tmp/testsubs:lathe subs:mills subs` - Specifies a colon (:) separated list of up to 10 directories to be searched when single-file subroutines are specified in G-code. These directories are searched after searching `[DISPLAY]PROGRAM_PREFIX` (if it is specified) and before searching `[WIZARD]WIZARD_ROOT` (if specified). The paths are searched in the order that they are listed. The first matching subroutine file found in the search is used. Directories are specified relative to the current directory for the INI file or as absolute paths. The list must contain no intervening whitespace.
- `CENTER_ARC_RADIUS_TOLERANCE_INCH = n` Default 0.00005
- `CENTER_ARC_RADIUS_TOLERANCE_MM = n` Default 0.00127
- `USER_M_PATH = myfuncs:/tmp/mcodes:experimental mcodes` - Specifies a list of colon (:) separated directories for user defined functions. Directories are specified relative to the current directory for the INI file or as absolute paths. The list must contain no intervening whitespace.

A search is made for each possible user defined function, typically (M100-M199). The search order is:

1. `[DISPLAY]PROGRAM_PREFIX` (if specified)

2. If [DISPLAY]PROGRAM_PREFIX is not specified, search the default location: nc_files
3. Then search each directory in the list [RS274NGC]USER_M_PATH
The first executable M1xx found in the search is used for each M1xx.

Note

The maximum number of USER_M_PATH directories is defined at compile time (typ: `USER_DEFINED_FUNCTION_MAX_DIRS == 5`).

- `INI_VARS = 1` Default 1
Allows G-code programs to read values from the INI file using the format `#<_ini[section]name>`. See [G-code Parameters](#).
- `HAL_PIN_VARS = 1` Default 1
Allows G-code programs to read the values of HAL pins using the format `#<_hal[HAL item]>`. Variable access is read-only. See [G-code Parameters](#) for more details and an important caveat.
- `RETAIN_G43 = 0` Default 0
When set, you can turn on G43 after loading the first tool, and then not worry about it through the program. When you finally unload the last tool, G43 mode is canceled.
- `OWORD_NARGS = 0` Default 0
If this feature is enabled then a called subroutine can determine the number of actual positional parameters passed by inspecting the `#<n_args>` parameter.
- `NO_DOWNCASE_OWORD = 0` Default 0
Preserve case in O-word names within comments if set, enables reading of mixed-case HAL items in structured comments like (*debug, #<_hal[MixedCaseItem]*).
- `OWORD_WARNONLY = 0` Default 0
Warn rather than error in case of errors in O-word subroutines.
- `DISABLE_G92_PERSISTENCE = 0` Default 0 Allow to clear the G92 offset automatically when config start-up.
- `DISABLE_FANUC_STYLE_SUB = 0` Default 0 If there is reason to disable Fanuc subroutines set it to 1.

Note

The above six options were controlled by the *FEATURES* bitmask in versions of LinuxCNC prior to 2.8. This INI tag will no longer work.

For reference:

```
FEATURES & 0x1  -> RETAIN_G43
FEATURES & 0x2  -> OWORD_NARGS
FEATURES & 0x4  -> INI_VARS
FEATURES & 0x8  -> HAL_PIN_VARS
FEATURES & 0x10 -> NO_DOWNCASE_OWORD
FEATURES & 0x20 -> OWORD_WARNONLY
```

Note

[WIZARD]WIZARD_ROOT is a valid search path but the Wizard has not been fully implemented and the results of using it are unpredictable.

- `LOG_LEVEL = 0` Default 0 Specify the log_level (default: -1)
- `LOG_FILE = file-name.log` For specify the file used for log the data.

- *REMAP=M400 modalgroup=10 argspec=Pq ngc=myprocedure* See [Remap Extending G-code](#) chapter for details.
- *ON_ABORT_COMMAND=O <on_abort> call* See [Remap Extending G-code](#) chapter for details.

9.2.2.5 [EMCMOT] Section

This section is a custom section and is not used by LinuxCNC directly. Most configurations use values from this section to load the motion controller. For more information on the motion controller see the [Motion](#) section.

- *EMCMOT = motmod* - the motion controller name is typically used here.
- *BASE_PERIOD = 50000* - the *Base* task period in nanoseconds.
- *SERVO_PERIOD = 1000000* - This is the "Servo" task period in nanoseconds.
- *TRAJ_PERIOD = 100000* - This is the *Trajectory Planner* task period in nanoseconds.
- *COMM_TIMEOUT = 1.0* - Number of seconds to wait for Motion (the realtime part of the motion controller) to acknowledge receipt of messages from Task (the non-realtime part of the motion controller).
- *HOMEMOD = alternate_homing_module [home_parms=value]* The *HOMEMOD* variable is optional. If specified, use a specified (user-built) module instead of the default (*homemod*). Module parameters (*home_parms*) may be included if supported by the named module. The setting may be overridden from the command line using the *-m* option (`$ linuxcnc -h`)

9.2.2.6 [TASK] Section

- *TASK = milltask* - Specifies the name of the *task* executable. The *task* executable does various things, such as communicate with the UIs over NML, communicate with the realtime motion planner over non-HAL shared memory, and interpret G-code. Currently there is only one task executable that makes sense for 99.9% of users, *milltask*.
- *CYCLE_TIME = 0.010* - The period, in seconds, at which TASK will run. This parameter affects the polling interval when waiting for motion to complete, when executing a pause instruction, and when accepting a command from a user interface. There is usually no need to change this number.

9.2.2.7 [HAL] section

- *HALFILE = example.hal* - Execute the file *example.hal* at start up. If *HALFILE* is specified multiple times, the files are executed in the order they appear in the INI file. Almost all configurations will have at least one *HALFILE*, and stepper systems typically have two such files, one which specifies the generic stepper configuration (*core_stepper.hal*) and one which specifies the machine pin out (*xxx_pinout.hal*).

HALFILES are found using a search.

If the named file is found in the directory containing the INI file, it is used.

If the named file is not found in this INI file directory, a search is made \leftrightarrow using a system library of HAL files.

If LinuxCNC is started with the `linuxcnc` script using the `"-H dirname"` option, the specified `dirname` is prepended to the search described above so that `"dirname"` is searched first. The `"-H dirname"` option may be specified more than once, directories are prepended in order.

A HALFILE may also be specified as an absolute path (when the name starts with a '/' character). Absolute paths are not recommended as their use may limit relocation of configurations.

- *HALFILE = texample.tcl [arg1 [arg2] ...]* - Execute the tcl file *texample.tcl* at start up with *arg1*, *arg2*, etc as *::argv* list. Files with a *.tcl* suffix are processed as above but use *haltcl* for processing. See the [HALTCL Chapter](#) for more information.
- *HALFILE = LIB:sys_example.hal* - Execute the system library file *sys_example.hal* at start up. Explicit use of the LIB: prefix causes use of the system library HALFILE without searching the INI file directory.
- *HALFILE = LIB:sys_texample.tcl [arg1 [arg2 ...]]* - Execute the system library file *sys_texample.tcl* at start up. Explicit use of the LIB: prefix causes use of the system library HALFILE without searching the INI file directory.

HALFILE items specify files that loadrt HAL components and make signal connections between component pins. Common mistakes are 1) omission of the *addf* statement needed to add a component's function(s) to a thread, 2) incomplete signal (net) specifiers. Omission of required *addf* statements is almost always an error. Signals usually include one or more input connections and a single output (but both are not strictly required). A system library file is provided to make checks for these conditions and report to stdout and in a popup GUI:

```
HALFILE = LIB:halcheck.tcl [nopopup]
```

Note

The LIB:halcheck.tcl line should be the last [HAL]HALFILE. Specify the *nopopup* option to suppress the popup message and allow immediate starting. Connections made using a POSTGUI_HALFILE are not checked.

- *TWOPASS = ON* - Use twopass processing for loading HAL components. With TWOPASS processing, [HAL]HALFILE= lines are processed in two passes. In the first pass (pass0), all HALFILES are read and multiple appearances of *loadrt* and *loadusr* commands are accumulated. These accumulated load commands are executed at the end of pass0. This accumulation allows load lines to be specified more than once for a given component (provided the names= names used are unique on each use). In the second pass (pass1), the HALFILES are reread and all commands except the previously executed load commands are executed.
- *TWOPASS = nodelete verbose* - The TWOPASS feature can be activated with any non-null string including the keywords *verbose* and *nodelete*. The *verbose* keyword causes printing of details to stdout. The *nodelete* keyword preserves temporary files in /tmp.

For more information see the [HAL TWOPASS](#) chapter.

- *HALCMD = command* - Execute *command* as a single HAL command. If *HALCMD* is specified multiple times, the commands are executed in the order they appear in the INI file. *HALCMD* lines are executed after all *HALFILE* lines.
 - *SHUTDOWN = shutdown.hal* - Execute the file *shutdown.hal* when LinuxCNC is exiting. Depending on the hardware drivers used, this may make it possible to set outputs to defined values when LinuxCNC is exited normally. However, because there is no guarantee this file will be executed (for instance, in the case of a computer crash) it is not a replacement for a proper physical e-stop chain or other protections against software failure.
-

- *POSTGUI_HALFILE* = *example2.hal* - Execute *example2.hal* after the GUI has created its HAL pins. Some GUIs create HAL pins and support the use of a postgui halfile to use them. GUIs that support postgui HAL files include Touchy, Axis, Gscreen, and gmoccapy. See section [pyVCP with Axis](#) for more information.
- *HALUI* = *halui* - adds the HAL user interface pins. For more information see the [HAL User Interface](#) chapter.

9.2.2.8 [HALUI] section

- *MDI_COMMAND* = *G53 G0 X0 Y0 Z0* - An MDI command can be executed by using *halui.mdi-command-00*. Increment the number for each command listed in the [HALUI] section.

9.2.2.9 [APPLICATIONS] Section

LinuxCNC can start other applications before the specified GUI is started. The applications can be started after a specified delay to allow for GUI-dependent actions (like creating GUI-specific HAL pins).

- *DELAY* = *value* - seconds to wait before starting other applications. A delay may be needed if an application has dependencies on [HAL]POSTGUI_HALFILE actions or GUI-created HAL pins (default DELAY=0).
- *APP* = *appname [arg1 [arg2 ...]]* - Application to be started. This specification can be included multiple times. The appname can be explicitly named as an absolute or tilde specified filename (first character is / or ~), a relative filename (first characters of filename are ./), or as a file in the inifile directory. If no executable file is found using these names, then the user search PATH is used to find the application.

Examples:

- Simulate inputs to HAL pins for testing (using *sim_pin* — a simple GUI to set inputs to parameters, unconnected pins, or signals with no writers):

```
APP = sim_pin motion.probe-input halui.abort motion.analog-in-00
```

- Invoke *halshow* with a previously saved watchlist. Since linuxcnc sets the working directory to the directory for the INI file, you can refer to files in that directory (example: *my.halshow*):

```
APP = halshow my.halshow
```

- Alternatively, a watchlist file identified with a full pathname could be specified:

```
APP = halshow ~/saved_shows/spindle.halshow
```

- Open *halscope* using a previously saved configuration:

```
APP = halscope -i my.halscope
```

9.2.2.10 [TRAJ] Section

Warning



The new Trajectory Planner (TP) is on by default. If you have no TP settings in your [TRAJ] section - LinuxCNC defaults to:

```
ARC_BLEND_ENABLE = 1
ARC_BLEND_FALLBACK_ENABLE = 0
ARC_BLEND_OPTIMIZATION_DEPTH = 50
ARC_BLEND_GAP_CYCLES = 4
ARC_BLEND_RAMP_FREQ = 100
```


The [TRAJ] section contains general parameters for the trajectory planning module in *motion*.

- `ARC_BLEND_ENABLE = 1` - Turn on new TP. If set to 0 TP uses parabolic blending (1 segment look ahead). Default value 1.
- `ARC_BLEND_FALLBACK_ENABLE = 0` - Optionally fall back to parabolic blends if the estimated speed is faster. However, this estimate is rough, and it seems that just disabling it gives better performance. Default value 0.
- `ARC_BLEND_OPTIMIZATION_DEPTH = 50` - Look ahead depth in number of segments.

To expand on this a bit, you can choose this value somewhat arbitrarily. Here's a formula to estimate how much *depth* you need for a particular config:

```
# n = v_max / (2.0 * a_max * t_c)
# where:
# n = optimization depth
# v_max = max axis velocity (UU / sec)
# a_max = max axis acceleration (UU / sec)
# t_c = servo period (seconds)
```

So, a machine with a maximum axis velocity of 10 IPS, a max acceleration of 100 IPS², and a servo period of 0.001 s would need:

$10 / (2.0 * 100 * 0.001) = 50$ segments to always reach maximum velocity along the fastest axis.

In practice, this number isn't that important to tune, since the look ahead rarely needs the full depth unless you have lots of very short segments. If during testing, you notice strange slowdowns and can't figure out where they come from, first try increasing this depth using the formula above.

If you still see strange slowdowns, it may be because you have short segments in the program. If this is the case, try adding a small tolerance for Naive CAM detection. A good rule of thumb is this:

```
# min_length ~= v_req * t_c
# where:
# v_req = desired velocity in UU / sec
# t_c = servo period (seconds)
```

If you want to travel along a path at 1 IPS = 60 IPM, and your servo period is 0.001 sec, then any segments shorter than `min_length` will slow the path down. If you set Naive CAM tolerance to around this min length, overly short segments will be combined together to eliminate this bottleneck. Of course, setting the tolerance too high means big path deviations, so you have to play with it a bit to find a good value. I'd start at 1/2 of the `min_length`, then work up as needed.

* `ARC_BLEND_GAP_CYCLES = 4` How short the previous segment must be before the trajectory planner *consumes* it.

Often, a circular arc blend will leave short line segments in between the blends. Since the geometry has to be circular, we can't blend over all of a line if the next one is a little shorter. Since the trajectory planner has to touch each segment at least once, it means that very tiny segments will slow things down significantly. My fix to this way to "consume" the short segment by making it a part of the blend arc. Since the line+blend is one segment, we don't have to slow down to hit the very short segment. Likely, you won't need to touch this setting.

* `ARC_BLEND_RAMP_FREQ = 20` - This is a *cutoff* frequency for using ramped velocity.

Ramped velocity in this case just means constant acceleration over the whole segment. This is less optimal than a trapezoidal velocity profile, since the acceleration is not maximized. However, if the segment is short enough, there isn't enough time to accelerate much before we hit the next segment. Recall the short line segments from the previous example. Since they're lines, there's no cornering acceleration, so we're free to accelerate up to the requested speed. However, if this line is between two arcs, then it will have to quickly decelerate again to be within the maximum speed of the next segment. This means that we have a spike of acceleration, then a spike of deceleration, causing a large jerk, for very little performance gain. This setting is a way to eliminate this jerk for short segments.

Basically, if a segment will complete in less time than $1 / \text{ARC_BLEND_RAMP_FREQ}$, we don't bother with a trapezoidal velocity profile on that segment, and use constant acceleration. (Setting $\text{ARC_BLEND_RAMP_FREQ} = 1000$ is equivalent to always using trapezoidal acceleration, if the servo loop is 1 kHz).

You can characterize the worst-case loss of performance by comparing the velocity that a trapezoidal profile reaches vs. the ramp:

```
# v_ripple = a_max / (4.0 * f)
# where:
# v_ripple = average velocity "loss" due to ramping
# a_max = max axis acceleration
# f = cutoff frequency from INI
```

For the aforementioned machine, the ripple for a 20 Hz cutoff frequency is $100 / (4 * 20) = 1.25$ IPS. This seems high, but keep in mind that it is only a worst-case estimate. In reality, the trapezoidal motion profile is limited by other factors, such as normal acceleration or requested velocity, and so the actual performance loss should be much smaller. Increasing the cutoff frequency can squeeze out more performance, but make the motion rougher due to acceleration discontinuities. A value in the range 20 Hz to 200 Hz should be reasonable to start.

Finally, no amount of tweaking will speed up a tool path with lots of small, tight corners, since you're limited by cornering acceleration.

- **SPINDLES = 3** - The number of spindles to support. It is imperative that this number matches the "num_spindles" parameter passed to the motion module.
- **COORDINATES = X Y Z** - The names of the axes being controlled. Only X, Y, Z, A, B, C, U, V, W are valid. Only axes named in **COORDINATES** are accepted in G-code. It is permitted to write an axis name more than once (e.g., X Y Y Z for a gantry machine). For the common *trivkins kinematics*, joint numbers are assigned in sequence according to the trivkins parameter *coordinates=*. So, for trivkins *coordinates=xz*, joint0 corresponds to X and joint1 corresponds to Z. See the kinematics man page (*\$ man kins*) for information on trivkins and other kinematics modules.
- **LINEAR_UNITS = <units>** - Specifies the *machine units* for linear axes. Possible choices are mm or inch. This does not affect the linear units in NC code (the G20 and G21 words do this).
- **ANGULAR_UNITS = <units>** - Specifies the *machine units* for rotational axes. Possible choices are *deg*, *degree* (360 per circle), *rad*, *radian* (2π per circle), *grad*, or *gon* (400 per circle). This does not affect the angular units of NC code. In RS274NGC, A-, B- and C- words are always expressed in degrees.
- **DEFAULT_LINEAR_VELOCITY = 0.0167** - The initial rate for jogs of linear axes, in machine units per second. The value shown in *Axis* equals machine units per minute.
- **DEFAULT_LINEAR_ACCELERATION = 2.0** - In machines with nontrivial kinematics, the acceleration used for "teleop" (Cartesian space) jogs, in *machine units* per second per second.
- **MAX_LINEAR_VELOCITY = 5.0** - The maximum velocity for any axis or coordinated move, in *machine units* per second. The value shown equals 300 units per minute.
- **MAX_LINEAR_ACCELERATION = 20.0** - The maximum acceleration for any axis or coordinated axis move, in *machine units* per second per second.
- **POSITION_FILE = position.txt** - If set to a non-empty value, the joint positions are stored between runs in this file. This allows the machine to start with the same coordinates it had on shutdown. This assumes there was no movement of the machine while powered off. If unset, joint positions are not stored and will begin at 0 each time LinuxCNC is started. This can help on smaller machines without home switches. If using the Mesa resolver interface this file can be used to emulate absolute encoders and eliminate the need for homing (with no loss of accuracy). See the hostmot2 manpage for more details.

- **`NO_FORCE_HOMING = 1`** - The default behavior is for LinuxCNC to force the user to home the machine before any MDI command or a program is run. Normally, only jogging is allowed before homing. For configurations using identity kinematics, setting `NO_FORCE_HOMING = 1` allows the user to make MDI moves and run programs without homing the machine first. Interfaces using identity kinematics without homing ability will need to have this option set to 1.

**Warning**

LinuxCNC will not know your joint travel limits when using `NO_FORCE_HOMING = 1`.

- **`HOME = 0 0 0 0 0 0 0 0`** - World home position needed for kinematics modules that compute world coordinates using `kinematicsForward()` when switching from joint to teleop mode. Up to nine coordinate values (X Y Z A B C U V W) may be specified, unused trailing items may be omitted. This value is only used for machines with nontrivial kinematics. On machines with trivial kinematics (mill, lathe, gantry types) this value is ignored. Note: the sim hexapod config requires a non-zero value for the Z coordinate.
- **`TPMOD = alternate_trajectory_planning_module`** [`tp_parms=value`]
The `TPMOD` variable is optional. If specified, use a specified (user-built) module instead of the default (`tpmod`). Module parameters (`tp_parms`) may be included if supported by the named module. The setting may be overridden from the command line using the `-t` option (`$ linuxcnc -h`).
- **`NO_PROBE_JOG_ERROR = 0`** - Allow to bypass probe tripped check when you jog manually.
- **`NO_PROBE_HOME_ERROR = 0`** - Allow to bypass probe tripped check when homing is in progress.

9.2.2.11 [KINS] Section

- **`JOINTS = 3`** - Specifies the number of joints (motors) in the system. For example, a `trivkins XYZ` machine with a single motor for each axis has 3 joints. A gantry machine with one motor on each of two of the axes, and two motors on the third axis, has 4 joints. (This config variable may be used by a GUI to set the number of joints (`num_joints`) specified to the motion module (`motmod`)).
- **`KINEMATICS = trivkins`** - Specify a kinematics module for the motion module. Guis may use this variable to specify the loadrt line in HAL files for the `motmod` module. For more information on kinematics modules see the manpage: `$ man kins`

9.2.2.12 [AXIS_<letter>] Section

The `<letter>` specifies one of: X Y Z A B C U V W

- **`MAX_VELOCITY = 1.2`** - Maximum velocity for this axis in [machine units](#) per second.
 - **`MAX_ACCELERATION = 20.0`** - Maximum acceleration for this axis in machine units per second squared.
 - **`MIN_LIMIT = -1000`** - The minimum limit (soft limit) for axis motion, in machine units. When this limit is exceeded, the controller aborts axis motion. The axis must be homed before `MIN_LIMIT` is in force. For a rotary axis (A,B,C typ) with unlimited rotation having no `MIN_LIMIT` for that axis in the `[AXIS_<letter>]` section a value of `-1e99` is used.
 - **`MAX_LIMIT = 1000`** - The maximum limit (soft limit) for axis motion, in machine units. When this limit is exceeded, the controller aborts axis motion. The axis must be homed before `MAX_LIMIT` is in force. For a rotary axis (A,B,C typ) with unlimited rotation having no `MAX_LIMIT` for that axis in the `[AXIS_<letter>]` section a value of `1e99` is used.
-

- *WRAPPED_ROTARY* = 1 - When this is set to 1 for an ANGULAR axis the axis will move 0-359.999 degrees. Positive Numbers will move the axis in a positive direction and negative numbers will move the axis in the negative direction.
- *LOCKING_INDEXER_JOINT* = 4 - This value selects a joint to use for a locking indexer for the specified axis <letter>. In this example, the joint is 4 which would correspond to the B axis for a XYZAB system with trivkins (identity) kinematics. When set, a G0 move for this axis will initiate an unlock with the joint.4.unlock pin then wait for the joint.4.is-unlocked pin then move the joint at the rapid rate for that joint. After the move the joint.4.unlock will be false and motion will wait for joint.4.is-unlocked to go false. Moving with other joints is not allowed when moving a locked rotary joint. To create the unlock pins, use the motmod parameter:

```
unlock_joints_mask=jointmask
```

The jointmask bits are: (LSB)0:joint0, 1:joint1, 2:joint2, ...

Example: loadrt motmod ... unlock_joints_mask=0x38 creates unlock pins for joints 3,4,5 *
OFFSET_AV_RATIO = 0.1 - If nonzero, this item enables the use of HAL input pins for external axis offsets:

```
axis.<letter>.eoffset-enable
axis.<letter>.eoffset-count
axis.<letter>.eoffset-scale
```

See the chapter: [External Axis Offsets](#) for usage information.

9.2.2.13 [JOINT_<num>] Sections

The <num> specifies the joint number 0 ... (num_joints-1) The value of *num_joints* is set by [KINS]JOINTS= The [JOINT_0], [JOINT_1], etc. sections contains general parameters for the individual components in the joint control module. The joint section names begin numbering at 0, and run through the number of joints specified in the [KINS]JOINTS entry minus 1.

Typically (for systems using *trivkins kinematics*, there is a 1:1 correspondence between a joint and an axis coordinate letter):

- JOINT_0 = X
- JOINT_1 = Y
- JOINT_2 = Z
- JOINT_3 = A
- JOINT_4 = B
- JOINT_5 = C
- JOINT_6 = U
- JOINT_7 = V
- JOINT_8 = W

Other kinematics modules with identity kinematics are available to support configurations with partial sets of axes. For example, using trivkins with coordinates=XZ, the joint-axes relationships are:

- JOINT_0 = X
- JOINT_1 = Z

For more information on kinematics modules see the manpage: `$ man kins`

- *TYPE* = *LINEAR* - The type of joint, either LINEAR or ANGULAR.
- *UNITS* = *INCH* - If specified, this setting overrides the related [TRAJ] UNITS setting. (e.g., [TRAJ]LINEAR if the TYPE of this joint is LINEAR, [TRAJ]ANGULAR_UNITS if the TYPE of this joint is ANGULAR)
- *MAX_VELOCITY* = *1.2* - Maximum velocity for this joint in [machine units](#) per second.
- *MAX_ACCELERATION* = *20.0* - Maximum acceleration for this joint in machine units per second squared.
- *BACKLASH* = *0.0000* - Backlash in machine units. Backlash compensation value can be used to make up for small deficiencies in the hardware used to drive an joint. If backlash is added to an joint and you are using steppers the STEPGEN_MAXACCEL must be increased to 1.5 to 2 times the MAX_ACCELERATION for the joint. Excessive backlash compensation can cause an joint to jerk as it changes direction. If a COMP_FILE is specified for a joint BACKLASH is not used.
- *COMP_FILE* = *file.extension* - The compensation file consists of map of position information for the joint. Compensation file values are in machine units. Each set of values are on one line separated by a space. The first value is the nominal value (the commanded position). The second and third values depend on the setting of COMP_FILE_TYPE. Points in between nominal values are interpolated between the two nominals. Compensation files must start with the smallest nominal and be in ascending order to the largest value of nominals. File names are case sensitive and can contain letters and/or numbers. Currently the limit inside LinuxCNC is for 256 triplets per joint. If COMP_FILE is specified for a joint, BACKLASH is not used.
- *COMP_FILE_TYPE* = *0 or 1* - Specifies the type of compensation file. The first value is the nominal (commanded) position for both types.
A *COMP_FILE_TYPE* must be specified for each *COMP_FILE*.
 - *Type 0*: The second value specifies the actual position as the joint is moving in the positive direction (increasing value) and the third value specifies the actual position as the joint is moving in the negative direction (decreasing value).

Type 0 Example

```
-1.000 -1.005 -0.995
0.000 0.002 -0.003
1.000 1.003 0.998
```

- *Type 1*: The second value specifies positive offset from nominal while traveling in the positive direction. The third value specifies the negative offset from nominal while traveling in a negative direction.

Type 1 Example

```
-1.000 0.005 -0.005
0.000 0.002 -0.003
1.000 0.003 -0.004
```

- *MIN_LIMIT* = *-1000* - The minimum limit for joint motion, in machine units. When this limit is reached, the controller aborts joint motion. For a rotary joint with unlimited rotation having no MIN_LIMIT for that joint in the [JOINT_N] section a the value -1e99 is used.
- *MAX_LIMIT* = *1000* - The maximum limit for joint motion, in machine units. When this limit is reached, the controller aborts joint motion. For a rotary joint with unlimited rotation having no MAX_LIMIT for that joint in the [JOINT_N] section a the value 1e99 is used.

Note

For **identity** kinematics, the [JOINT_N]MIN_LIMIT,MAX_LIMIT settings must equal or exceed the corresponding (one-to-one identity) [AXIS_L] limits. These settings are verified at startup when the trivkins kinematics modules is specified.

Note

The [JOINT_N]MIN_LIMIT, MAX_LIMIT settings are enforced while jogging in joint mode prior to homing. After homing, [AXIS_L]MIN_LIMIT,MAX_LIMIT coordinate limits are used as constraints for axis (coordinate letter) jogging and by the trajectory planning used for G-code moves (programs and MDI commands). The trajectory planner works in Cartesian space (XYZABCUVW) and has no information about the motion of joints implemented by **any** kinematics module. It is possible for joint limit violations to occur for G-code that obeys trajectory planning position limits when non identity kinematics are used. The motion module always detects joint position limit violations and faults if they occur during the execution of G-code commands. See also related github issue #97.

- *MIN_FERROR = 0.010* - This is the value in machine units by which the joint is permitted to deviate from commanded position at very low speeds. If MIN_FERROR is smaller than FERROR, the two produce a ramp of error trip points. You could think of this as a graph where one dimension is speed and the other is permitted following error. As speed increases the amount of following error also increases toward the FERROR value.
- *FERROR = 1.0* - FERROR is the maximum allowable following error, in machine units. If the difference between commanded and sensed position exceeds this amount, the controller disables servo calculations, sets all the outputs to 0.0, and disables the amplifiers. If MIN_FERROR is present in the .ini file, velocity-proportional following errors are used. Here, the maximum allowable following error is proportional to the speed, with FERROR applying to the rapid rate set by [TRAJ]MAX_VELOCITY, and proportionally smaller following errors for slower speeds. The maximum allowable following error will always be greater than MIN_FERROR. This prevents small following errors for stationary axes from inadvertently aborting motion. Small following errors will always be present due to vibration, etc.
- *LOCKING_INDEXER = 1* - Indicates the joint is used as a locking indexer.

These parameters are Homing related, for a better explanation read the [Homing Configuration](#) Chapter.

- *HOME = 0.0* - The position that the joint will go to upon completion of the homing sequence.
- *HOME_OFFSET = 0.0* - The joint position of the home switch or index pulse, in [machine units](#). When the home point is found during the homing process, this is the position that is assigned to that point. When sharing home and limit switches and using a home sequence that will leave the home/limit switch in the toggled state the home offset can be used define the home switch position to be other than 0 if your HOME position is desired to be 0.
- *HOME_SEARCH_VEL = 0.0* - Initial homing velocity in machine units per second. Sign denotes direction of travel. A value of zero means assume that the current location is the home position for the machine. If your machine has no home switches you will want to leave this value at zero.
- *HOME_LATCH_VEL = 0.0* - Homing velocity in machine units per second to the home switch latch position. Sign denotes direction of travel.
- *HOME_FINAL_VEL = 0.0* - Velocity in machine units per second from home latch position to home position. If left at 0 or not included in the joint rapid velocity is used. Must be a positive number.
- *HOME_USE_INDEX = NO* - If the encoder used for this joint has an index pulse, and the motion card has provision for this signal you may set it to yes. When it is yes, it will affect the kind of home pattern used. Currently, you can't home to index with steppers unless you're using stepgen in velocity mode and PID.

- `HOME_INDEX_NO_ENCODER_RESET = NO` - Use YES if the encoder used for this joint does not reset its counter when an index pulse is detected after assertion of the joint `index_enable` HAL pin. Applicable only for `HOME_USE_INDEX = YES`.
- `HOME_IGNORE_LIMITS = NO` - When you use the limit switch as a home switch and the limit switch this should be set to YES. When set to YES the limit switch for this joint is ignored when homing. You must configure your homing so that at the end of your home move the home/limit switch is not in the toggled state you will get a limit switch error after the home move.
- `HOME_IS_SHARED = <n>` - If the home input is shared by more than one joint set `<n>` to 1 to prevent homing from starting if the one of the shared switches is already closed. Set `<n>` to 0 to permit homing if a switch is closed.
- `HOME_ABSOLUTE_ENCODER = 0 | 1 | 2` - Used to indicate the joint uses an absolute encoder. At a request for homing, the current joint value is set to the `HOME_OFFSET` value. If the `HOME_ABSOLUTE_ENCODER` setting is 1, the machine makes the usual final move to the `HOME` value. If the `HOME_ABSOLUTE_ENCODER` setting is 2, no final move is made.
- `HOME_SEQUENCE = <n>` - Used to define the "Home All" sequence. `<n>` must start at 0 or 1 or -1. Additional sequences may be specified with numbers increasing by 1 (in absolute value). Skipping of sequence numbers is not allowed. If a `HOME_SEQUENCE` is omitted, the joint will not be homed by the "Home All" function. More than one joint can be homed at the same time by specifying the same sequence number for more than one joint. A negative sequence number is used to defer the final move for all joints having that (negative or positive) sequence number. For additional info, see: [HOME SEQUENCE](#).
- `VOLATILE_HOME = 0` - When enabled (set to 1) this joint will be unhomed if the Machine Power is off or if E-Stop is on. This is useful if your machine has home switches and does not have position feedback such as a step and direction driven machine.

These parameters are relevant to joints controlled by servos.



Warning

The following are custom INI file entries that you may find in a sample INI file or a wizard generated file. These are not used by the LinuxCNC software. They are only there to put all the settings in one place. For more information on custom INI file entries see the [Custom Sections and Variables](#) subsection.

The following items might be used by a PID component and the assumption is that the output is volts.

- `DEADBAND = 0.000015` - How close is close enough to consider the motor in position, in [machine units](#). This is often set to a distance equivalent to 1, 1.5, 2, or 3 encoder counts, but there are no strict rules. Looser (larger) settings allow less servo *hunting* at the expense of lower accuracy. Tighter (smaller) settings attempt higher accuracy at the expense of more servo *hunting*. Is it really more accurate if it's also more uncertain? As a general rule, it's good to avoid, or at least limit, servo *hunting* if you can.

Be careful about going below 1 encoder count, since you may create a condition where there is no place that your servo is happy. This can go beyond *hunting* (slow) to *nervous* (rapid), and even to *squealing* which is easy to confuse with oscillation caused by improper tuning. Better to be a count or two loose here at first, until you've been through *gross tuning* at least.

Example of calculating machine units per encoder pulse to use in deciding DEADBAND value:

$$\frac{1 \text{ revolution}}{1000 \text{ lines}} \times \frac{1 \text{ line}}{4 \text{ pulse/line}} \times \frac{0.2 \text{ units}}{1 \text{ revolution}} = \frac{0.200 \text{ units}}{4000 \text{ pulses}} = \frac{0.00005 \text{ units}}{1 \text{ pulse}}$$

- $BIAS = 0.000$ - This is used by hm2-servo and some others. Bias is a constant amount that is added to the output. In most cases it should be left at zero. However, it can sometimes be useful to compensate for offsets in servo amplifiers, or to balance the weight of an object that moves vertically. Bias is turned off when the PID loop is disabled, just like all other components of the output.
- $P = 50$ - The proportional gain for the joint servo. This value multiplies the error between commanded and actual position in machine units, resulting in a contribution to the computed voltage $\frac{\text{volts}}{\text{unit}}$ for the motor amplifier. The units on the P gain are volts per machine unit, e.g., $\frac{\text{volts}}{\text{unit}}$.
- $I = 0$ - The integral gain for the joint servo. The value multiplies the cumulative error between commanded and actual position in machine units, resulting in a contribution to the computed voltage $\frac{\text{volts}}{\text{unit second}}$ for the motor amplifier. The units on the I gain are volts per machine unit second, e.g., $\frac{\text{volts}}{\text{unit second}}$.
- $D = 0$ - The derivative gain for the joint servo. The value multiplies the difference between the current and previous errors, resulting in a contribution to the computed voltage for the motor amplifier. The units on the D gain are volts per machine unit per second, e.g., $\frac{\text{volts}}{\text{unit second}}$.
- $FF0 = 0$ - The 0th order feed forward gain. This number is multiplied by the commanded position, resulting in a contribution to the computed voltage for the motor amplifier. The units on the FF0 gain are volts per machine unit, e.g., $\frac{\text{volts}}{\text{unit}}$.
- $FF1 = 0$ - The 1st order feed forward gain. This number is multiplied by the change in commanded position per second, resulting in a contribution to the computed voltage for the motor amplifier. The units on the FF1 gain are volts per machine unit per second, e.g., $\frac{\text{volts}}{\text{unit second}}$.
- $FF2 = 0$ - The 2nd order feed forward gain. This number is multiplied by the change in commanded position per second per second, resulting in a contribution to the computed voltage for the motor amplifier. The units on the FF2 gain are volts per machine unit per second per second, e.g., $\frac{\text{volts}}{\text{unit second}^2}$.
- $OUTPUT_SCALE = 1.000$
- $OUTPUT_OFFSET = 0.000$ - These two values are the scale and offset factors for the joint output to the motor amplifiers. The second value (offset) is subtracted from the computed output (in volts), and divided by the first value (scale factor), before being written to the D/A converters. The units on the scale value are in true volts per DAC output volts. The units on the offset value are in volts. These can be used to linearize a DAC. Specifically, when writing outputs, the LinuxCNC first converts the desired output in quasi-SI units to raw actuator values, e.g., volts for an amplifier DAC. This scaling looks like: $raw = \frac{output - offset}{scale}$

The value for scale can be obtained analytically by doing a unit analysis, i.e., units are [output SI units]/[actuator units]. For example, on a machine with a velocity mode amplifier such that 1 V results in 250 mm/s velocity.

$$amplifier[\text{volts}] = (output[\frac{\text{mm}}{\text{sec}}] - offset[\frac{\text{mm}}{\text{sec}}]) / 250 \frac{\text{mm}}{\text{secvolt}}$$

Note that the units of the offset are in machine units, e.g., mm/sec, and they are pre-subtracted from the sensor readings. The value for this offset is obtained by finding the value of your output which yields 0.0 for the actuator output. If the DAC is linearized, this offset is normally 0.0.

The scale and offset can be used to linearize the DAC as well, resulting in values that reflect the combined effects of amplifier gain, DAC non-linearity, DAC units, etc.

To do this, follow this procedure.

1. Build a calibration table for the output, driving the DAC with a desired voltage and measuring the result.
2. Do a least-squares linear fit to get coefficients a, b such that $measured = a * raw + b$
3. Note that we want raw output such that our measured result is identical to the commanded output. This means
 - a. $command = a * raw + b$
 - b. $raw = (command - b) / a$
4. As a result, the a and b coefficients from the linear fit can be used as the scale and offset for the controller directly.

See the following table for an example of voltage measurements.

Table 9.1: Output Voltage Measurements

Raw	Measured
-10	-9.93
-9	-8.83
0	-0.03
1	0.96
9	9.87
10	10.87

- $MAX_OUTPUT = 10$ - The maximum value for the output of the PID compensation that is written to the motor amplifier, in volts. The computed output value is clamped to this limit. The limit is applied before scaling to raw output units. The value is applied symmetrically to both the plus and the minus side.
- $INPUT_SCALE = 20000$ - in Sample configs
- $ENCODER_SCALE = 20000$ - in PnCconf built configs

Specifies the number of pulses that corresponds to a move of one machine unit as set in the [TRAJ] section. For a linear joint one machine unit will be equal to the setting of LINEAR_UNITS. For an angular joint one unit is equal to the setting in ANGULAR_UNITS. A second number, if specified, is ignored. For example, on a 2000 counts per rev encoder, and 10 revs/inch gearing, and desired units of inch, we have:

$$input\ scale = 2000 \frac{counts}{rev} * 10 \frac{rev}{inch} = 20000 \frac{counts}{inch}$$

These parameters are relevant to joints controlled by steppers.



Warning

The following are custom INI file entries that you may find in a sample INI file or a wizard generated file. These are not used by the LinuxCNC software. They are only there to put all the settings in one place. For more information on custom INI file entries see the [Custom Sections and Variables](#) subsection.

The following items might be used by a stepgen component.

- *SCALE* = 4000 - in Sample configs
- *STEP_SCALE* = 4000 - in PnCconf built configs

Specifies the number of pulses that corresponds to a move of one machine unit as set in the [TRAJ] section. For stepper systems, this is the number of step pulses issued per machine unit. For a linear joint one machine unit will be equal to the setting of LINEAR_UNITS. For an angular joint one unit is equal to the setting in ANGULAR_UNITS. For servo systems, this is the number of feedback pulses per machine unit. A second number, if specified, is ignored.

For example, on a 1.8 degree stepper motor with half-stepping, and 10 revs/inch gearing, and desired machine units of inch, we have:

$$\text{input scale} = \frac{2 \text{ steps}}{1.8 \text{ degrees}} * 360 \frac{\text{degree}}{\text{rev}} * 10 \frac{\text{rev}}{\text{inch}} = 4000 \frac{\text{steps}}{\text{inch}}$$

Note

Old INI and HAL files used INPUT_SCALE for this value.

- *ENCODER_SCALE* = 20000 (Optionally used in PNCconf built configs) - Specifies the number of pulses that corresponds to a move of one machine unit as set in the [TRAJ] section. For a linear joint one machine unit will be equal to the setting of LINEAR_UNITS. For an angular joint one unit is equal to the setting in ANGULAR_UNITS. A second number, if specified, is ignored. For example, on a 2000 counts per rev encoder, and 10 revs/inch gearing, and desired units of inch, we have:

$$\text{input scale} = 2000 \frac{\text{counts}}{\text{rev}} * 10 \frac{\text{rev}}{\text{inch}} = 20000 \frac{\text{counts}}{\text{inch}}$$

- *STEPGEN_MAXACCEL* = 21.0 - Acceleration limit for the step generator. This should be 1% to 10% larger than the joint MAX_ACCELERATION. This value improves the tuning of stepgen's "position loop". If you have added backlash compensation to an joint then this should be 1.5 to 2 times greater than MAX_ACCELERATION.
- *STEPGEN_MAXVEL* = 1.4 - Older configuration files have a velocity limit for the step generator as well. If specified, it should also be 1% to 10% larger than the joint MAX_VELOCITY. Subsequent testing has shown that use of STEPGEN_MAXVEL does not improve the tuning of stepgen's position loop.

9.2.2.14 [SPINDLE_<num>] Section(s)

The <num> specifies the spindle number 0 ... (num_spindles-1) The value of *num_spindles* is set by [TRAJ]SPINDLES=

- *MAX_VELOCITY* = 20000 The maximum spindle speed (in rpm) for the specified spindle. Optional.
 - *MIN_VELOCITY* = 3000 The minimum spindle speed (in rpm) for the specified spindle. Optional. Many spindles have a minimum speed below which they should not be run. Any spindle speed command below this limit will be /increased/ to this limit.
 - *MAX_REVERSE_VELOCITY* = 20000 This setting will default to MAX_VELOCITY if omitted. It can be used in cases where the spindle speed is limited in reverse. Set to zero for spindles which must not be run in reverse. In this context "max" refers to the absolute magnitude of the spindle speed.
-

- *MIN_REVERSE_VELOCITY = 3000* This setting is equivalent to *MIN_VELOCITY* but for reverse spindle rotation. It will default to the *MIN_VELOCITY* if omitted.
- *INCREMENT = 200* Sets the step size for spindle speed increment / decrement commands. This can have a different value for each spindle. This setting is effective with Axis and Touchy but note that some GUIs may handle things differently.
- *HOME_SEARCH_VELOCITY = 100* - FIXME: Spindle homing not yet working Sets the homing speed (rpm) for the spindle. The spindle will rotate at this velocity during the homing sequence until the spindle index is located, at which point the spindle position will be set to zero. Note that it makes no sense for the spindle home position to be any value other than zero, and so there is no provision to do so.
- *HOME_SEQUENCE = 0* - FIXME: Spindle homing not yet working Controls where in the general homing sequence the spindle homing rotations occur. Set the *HOME_SEARCH_VELOCITY* to zero to avoid spindle rotation during the homing sequence.

9.2.2.15 [EMCIO] Section

- *EMCIO = io* - Name of IO controller program.
- *CYCLE_TIME = 0.100* - The period, in seconds, at which EMCIO will run. Making it 0.0 or a negative number will tell EMCIO not to sleep at all. There is usually no need to change this number.
- *TOOL_TABLE = tool.tbl* - The file which contains tool information, described in the User Manual.
- *DB_PROGRAM = db_program* - Path to an executable program that manages tool data. (When a *DB_PROGRAM* is specified, a *TOOL_TABLE* entry is ignored)
- *TOOL_CHANGE_POSITION = 0 0 2* - Specifies the XYZ location to move to when performing a tool change if three digits are used. Specifies the XYZABC location when 6 digits are used. Specifies the XYZABCUVW location when 9 digits are used. Tool Changes can be combined. For example if you combine the quill up with change position you can move the Z first then the X and Y.
- *TOOL_CHANGE_WITH_SPINDLE_ON = 1* - The spindle will be left on during the tool change when the value is 1. Useful for lathes or machines where the material is in the spindle, not the tool.
- *TOOL_CHANGE_QUILL_UP = 1* - The Z axis will be moved to machine zero prior to the tool change when the value is 1. This is the same as issuing a G0 G53 Z0.
- *TOOL_CHANGE_AT_G30 = 1* - The machine is moved to reference point defined by parameters 5181-5186 for G30 if the value is 1. For more information see [G-code Parameters](#) and [G-code G30-G30.1](#).
- *RANDOM_TOOLCHANGER = 1* - This is for machines that cannot place the tool back into the pocket it came from. For example, machines that exchange the tool in the active pocket with the tool in the spindle.

9.3 Homing Configuration

9.3.1 Overview

Homing sets the zero origin of the G53 machine coordinates. Soft limits are defined relative to the machine origin. The soft limits automatically decelerate and stop the axes before they hit the limits switches. A properly configured and functioning machine will not move beyond soft(ware) limits and will have the machine origin set as repeatable as the home switch/index mechanism is. Linuxcnc can be homed by eye (alignment marks), with switches, with switches and an encoder index, or by using

absolute encoders. Homing seems simple enough - just move each joint to a known location, and set LinuxCNC's internal variables accordingly. However, different machines have different requirements, and homing is actually quite complicated.

Note

While it is possible to use LinuxCNC without homing switches/home procedures or limit switches, it defeats the extra security of the soft limits.

9.3.2 Prerequisite

Homing relies on some fundamental machine assumptions.

- The negative and positive directions are based on [Tool Movement](#) which can be different than the actual machine movement. ie on a mill typically the table moves rather than the tool.
- Everything is referenced from the G53 machine zero origin, the origin can be anywhere (even outside where you can move)
- The G53 machine zero origin is typically inside the soft limits area but not necessarily.
- The homing switch offset sets where the origin is, but even it is referenced from the origin.
- When using encoder index homing, the home switch offset is calculated from the encoder reference position, after the home switch has been tripped.
- The negative soft(ware) limits are the most you can move in the negative direction after homing. (but they might not be negative in the absolute sense)
- The positive soft(ware) limits are the most you can move in the positive direction after homing. (but they might not be positive in the absolute sense, though it is usual to set it as a positive number)
- Soft(ware) limits are inside the limit switch area.
- (Final) Homed Position is inside the soft limit area
- (If using switch based homing) the homing switch(es) either utilize the limit switches (shared home / limit switch), or when using a separate home switch, are inside the limit switch area.
- If using a separate homing switch, it's possible to start homing on the wrong side of the home switch, which combined with HOME IGNORE LIMITS option will lead to a hard crash. You can avoid this by making the home switch toggle it's state when the trip dog is on a particular side until it returns passed the trip point again. Said another way, the home switch state must represent the position of the dog relative to the switch (ie *before* or *after* the switch), and must stay that way even if the dog coasts past the switch in the same direction.

Note

While it is possible to use LinuxCNC with the G53 machine origin outside the soft machine limits, if you use G28 or G30 without setting the parameters it goes to the origin by default. This would trip the limit switches before getting to position.

9.3.3 Separate Home Switch Example Layout

This example shows minimum and maximum limit switches with a separate home switch.

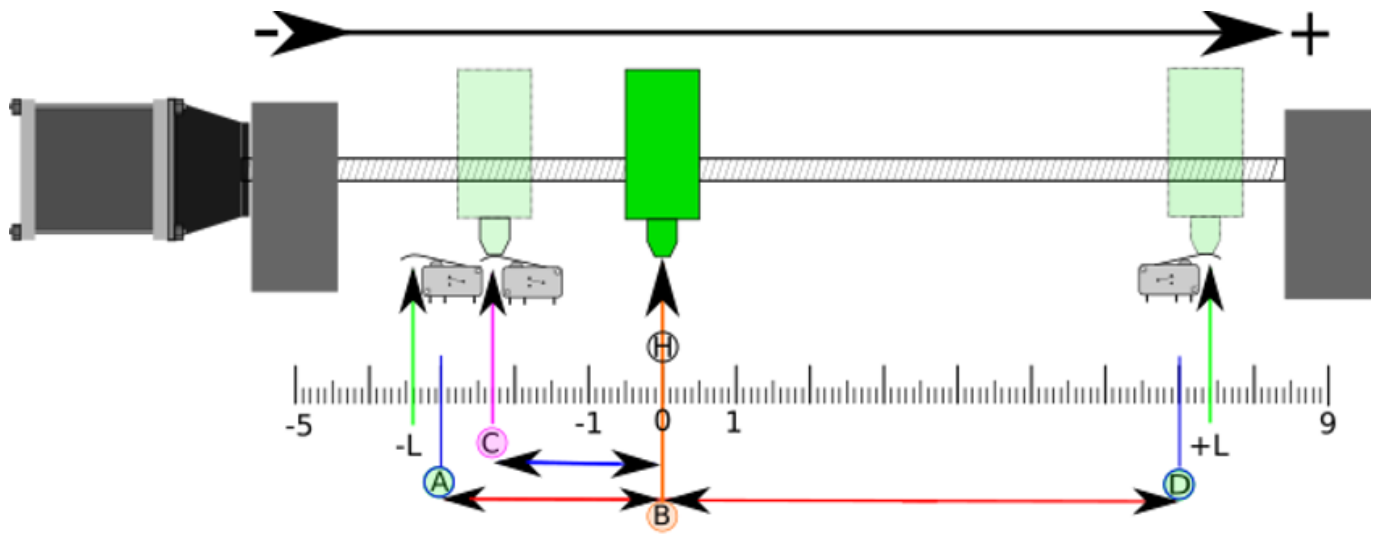


Figure 9.1: Demonstrative Separate Switch Layout

- A is the negative soft limit
- B is the G53 machine coordinate Origin
- C is the home switch trip point
- D is the positive soft limit
- H is the final home position (HOME) = 0 units
- The -L and +L are the limit switches trip points
- A<->B is the negative soft limits (MIN_LIMITS) = -3 units
- B<->C is the home_offset (HOME_OFFSET) = -2.3 units
- B<->D is the positive soft limits (MAX_LIMITS) = 7 units
- A<->D is the total travel = 10 units
- The distance between the limit switches and soft limits (-L<->A and D<->+L) is magnified in this example
- Note that there is distance between the limit switches and actual physical hard contact for coasting after the amplifier is disabled.

Note

Homing sets the G53 coordinate system, while the machine origin (zero point) can be anywhere, setting the zero point at the negative soft limit makes all G53 coordinates positive, which is probably easiest to remember. Do this by setting MIN_LIMIT = 0 and make sure MAX_LIMIT is positive.

9.3.4 Shared Limit/Home Switch Example Layout

This example shows a maximum limit switch and a combined minimum limit/home switch.

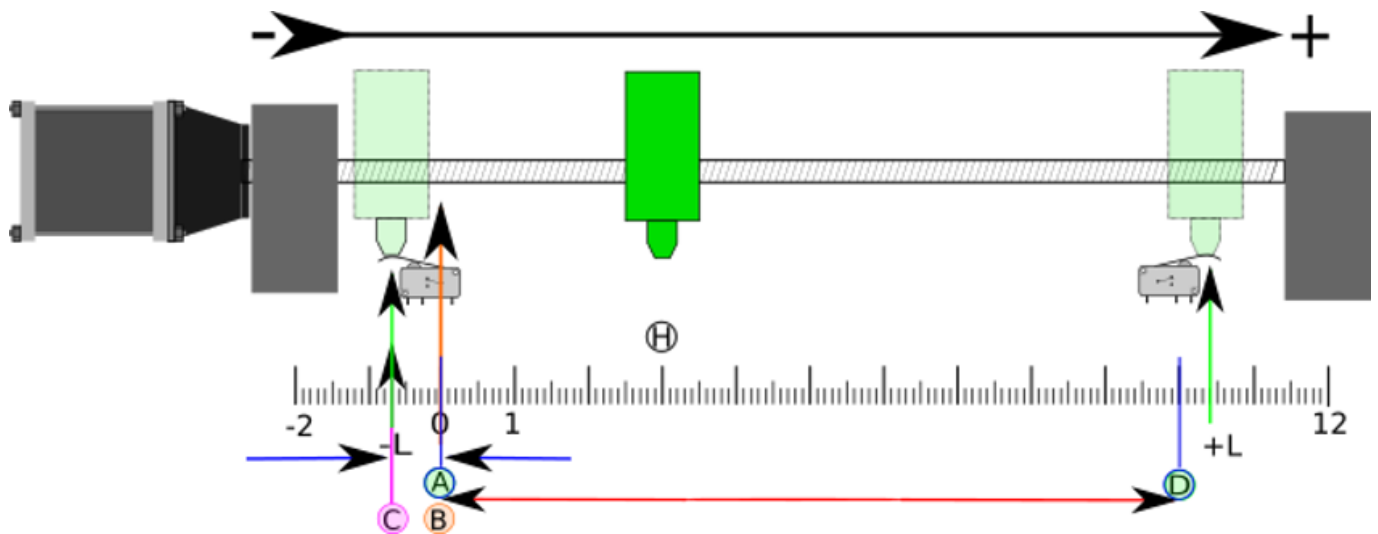


Figure 9.2: Demonstrative Shared Switch Layout

- A is the negative soft limit.
- B is the G53 machine coordinate Origin.
- C is the home switch trip point shared with (-L) minimum limit trip.
- D is the positive soft limit.
- H is the final home position (HOME) = 3 units.
- The -L and +L are the limit switch trip points.
- A<->B is the negative soft limits (MIN_LIMITS) = 0 units.
- B<->C is the home_offset (HOME_OFFSET) = -0.7 units.
- B<->D is the positive soft limits (MAX_LIMITS) 10 units.
- A<->D is the total travel = 10 units.
- The distance between the limits switches and soft limits (-L<->A and D<->+L) is magnified in this example.
- Note that there is distance between the limit switches and actual physical hard contact for coasting after the amplifier is disabled.

9.3.5 Homing Sequence

There are four possible homing sequences defined by the sign of HOME_SEARCH_VEL and HOME_LATCH_V along with the associated configuration parameters as shown in the following table. Two basic conditions exist, HOME_SEARCH_VEL and HOME_LATCH_V are the same sign or they are opposite signs. For a more detailed description of what each configuration parameter does, see the following section.



Figure 9.3: Homing Sequences

9.3.6 Configuration

The following determines exactly how the home sequence behaves. They are defined in an [JOINT_n] section of the INI file.

Homing Type	HOME_SEARCH_VEL	HOME_LATCH_VEL	HOME_USE_INDEX
Immediate	0	0	NO
Index-only	0	nonzero	YES
Switch-only	nonzero	nonzero	NO
Switch and Index	nonzero	nonzero	YES

Note

Any other combinations may result in an error.

9.3.6.1 HOME_SEARCH_VEL

This variable has units of machine-units per second.

The default value is zero. A value of zero causes LinuxCNC to assume that there is no home switch; the search stage of homing is skipped.

If HOME_SEARCH_VEL is non-zero, then LinuxCNC assumes that there is a home switch. It begins by checking whether the home switch is already tripped. If tripped it backs off the switch at HOME_SEARCH_VEL. The direction of the back-off is opposite the sign of HOME_SEARCH_VEL. Then it searches for the home switch by moving in the direction specified by the sign of HOME_SEARCH_VEL, at a speed determined by its absolute value. When the home switch is detected, the joint will stop as fast as possible, but there will always be some overshoot. The amount of overshoot depends on the speed. If it is too high, the joint might overshoot enough to hit a limit switch or crash into the end of travel. On the other hand, if HOME_SEARCH_VEL is too low, homing can take a long time.

9.3.6.2 HOME_LATCH_VEL

This variable has units of machine-units per second.

Specifies the speed and direction that LinuxCNC uses when it makes its final accurate determination of the home switch (if present) and index pulse location (if present). It will usually be slower than the search velocity to maximize accuracy. If HOME_SEARCH_VEL and HOME_LATCH_VEL have the same sign, then the latch phase is done while moving in the same direction as the search phase. (In that case, LinuxCNC first backs off the switch, before moving towards it again at the latch velocity.) If HOME_SEARCH_VEL and HOME_LATCH_VEL have opposite signs, the latch phase is done while moving in the opposite direction from the search phase. That means LinuxCNC will latch the first pulse after it moves off the switch. If HOME_SEARCH_VEL is zero (meaning there is no home switch), and this parameter is nonzero, LinuxCNC goes ahead to the index pulse search. If HOME_SEARCH_VEL is non-zero and this parameter is zero, it is an error and the homing operation will fail. The default value is zero.

9.3.6.3 HOME_FINAL_VEL

This variable has units of machine-units per second.

It specifies the speed that LinuxCNC uses when it makes its move from HOME_OFFSET to the HOME position. If the HOME_FINAL_VEL is missing from the INI file, then the maximum joint speed is used to make this move. The value must be a positive number.

9.3.6.4 HOME_IGNORE_LIMITS

Can hold the values YES / NO. The default value for this parameter is NO. This flag determines whether LinuxCNC will ignore the limit switch input for this joint while homing. This setting will not ignore limit inputs for other joints. If you do not have a separate home switch set this to YES and connect the limit switch signal to the joint home switch input in HAL. LinuxCNC will ignore the limit switch input for this joint while homing. To use only one input for all homing and limits you will have to block the limit signals of the joints not homing in HAL and home one joint at a time.

9.3.6.5 HOME_USE_INDEX

Specifies whether or not there is an index pulse. If the flag is true (HOME_USE_INDEX = YES), LinuxCNC will latch on the rising edge of the index pulse. If false, LinuxCNC will latch on either the rising or falling edge of the home switch (depending on the signs of HOME_SEARCH_VEL and HOME_LATCH_VEL). The default value is NO.

Note

HOME_USE_INDEX requires connections in your HAL file to joint.n.index-enable from the encoder.n.index-enable.

9.3.6.6 HOME_INDEX_NO_ENCODER_RESET

Default is NO. Use YES if the encoder used for this joint does not reset its counter when an index pulse is detected after assertion of the joint index_enable HAL pin. Applicable only for HOME_USE_INDEX = YES.

9.3.6.7 HOME_OFFSET

This defines the location of the origin zero point of the G53 machine coordinate system. It is the distance (offset), in joint units, from the machine origin to the home switch trip point or index pulse. After detecting the switch trip point/index pulse, LinuxCNC sets the joint coordinate position to HOME_OFFSET, thus defining the origin, which the soft limits references from. The default value is zero.

Note

The home switch location, as indicated by the HOME_OFFSET variable, can be inside or outside the soft limits. They will be shared with or inside the hard limit switches.

9.3.6.8 HOME

The position that the joint will go to upon completion of the homing sequence. After detecting the home switch or home switch then index pulse (depending on configuration), and setting the coordinate of that point to HOME_OFFSET, LinuxCNC makes a move to HOME as the final step of the homing process. The default value is zero. Note that even if this parameter is the same as HOME_OFFSET, the joint will slightly overshoot the latched position as it stops. Therefore there will always be a small move at this time (unless HOME_SEARCH_VEL is zero, and the entire search/latch stage was skipped). This final move will be made at the joint's maximum velocity unless HOME_FINAL_VEL has been set.

Note

The distinction between *HOME_OFFSET* and *HOME* is that *HOME_OFFSET* first establishes the origin location and scale on the machine by applying the *HOME_OFFSET* value to the location where home was found, and then *HOME* says where the joint should move to on that scale.

9.3.6.9 HOME_IS_SHARED

If there is not a separate home switch input for this joint, but a number of momentary switches wired to the same pin, set this value to 1 to prevent homing from starting if one of the shared switches is already closed. Set this value to 0 to permit homing even if the switch is already closed.

9.3.6.10 HOME_ABSOLUTE_ENCODER

Use for absolute encoders. When a request is made to home the joint, the current joint position is set to the *[JOINT_n]HOME_OFFSET* value.

The final move to the *[JOINT_n]HOME* position is optional according to the *HOME_ABSOLUTE_ENCODER* setting:

```
HOME_ABSOLUTE_ENCODER = 0 (Default) joint does not use an absolute encoder
HOME_ABSOLUTE_ENCODER = 1 Absolute encoder, final move to [JOINT_n]HOME
HOME_ABSOLUTE_ENCODER = 2 Absolute encoder, NO final move to [JOINT_n]HOME
```

Note

A *HOME_IS_SHARED* setting is silently ignored.

Note

A request to rehome the joint is silently ignored.

9.3.6.11 HOME_SEQUENCE

Used to define a multi-joint homing sequence **HOME ALL** and enforce homing order (e.g., Z may not be homed if X is not yet homed). A joint may be homed after all joints with a lower (absolute value) *HOME_SEQUENCE* have already been homed and are at the *HOME_OFFSET*. If two joints have the same *HOME_SEQUENCE*, they may be homed at the same time.

Note

If *HOME_SEQUENCE* is not specified then the joint will not be homed by the **HOME ALL** sequence (but may be homed by individual joint-specific homing commands).

The initial *HOME_SEQUENCE* number may be 0, 1 (or -1). The absolute value of sequence numbers must increment by one — skipping sequence numbers is not supported. If a sequence number is omitted, **HOME ALL** homing will stop upon completion of the last valid sequence number.

Negative *HOME_SEQUENCE* values indicate that joints in the sequence should **synchronize the final move** to *[JOINT_n]HOME* by waiting until all joints in the sequence are ready. If any joint has a **negative** *HOME_SEQUENCE* value, then all joints with the same absolute value (positive or negative) of the *HOME_SEQUENCE* item value will synchronize the final move.

A **negative** HOME_SEQUENCE also applies to commands to home a single joint. If the HOME_SEQUENCE value is **negative**, all joints having the same absolute value of that HOME_SEQUENCE will be **homed together with a synchronized final move**. If the HOME_SEQUENCE value is zero or positive, a command to home the joint will home only the specified joint.

Joint mode jogging of joints having a negative HOME_SEQUENCE is disallowed. In common gantry applications, such jogging can lead to misalignment (racking). Note that conventional jogging in world coordinates is always available once a machine is homed.

Examples for a 3 joint system

Two sequences (0,1), no synchronization

```
[JOINT_0]HOME_SEQUENCE = 0
[JOINT_1]HOME_SEQUENCE = 1
[JOINT_2]HOME_SEQUENCE = 1
```

Two sequences, joints 1 and 2 synchronized

```
[JOINT_0]HOME_SEQUENCE = 0
[JOINT_1]HOME_SEQUENCE = -1
[JOINT_2]HOME_SEQUENCE = -1
```

With mixed positive and negative values, joints 1 and 2 synchronized

```
[JOINT_0]HOME_SEQUENCE = 0
[JOINT_1]HOME_SEQUENCE = -1
[JOINT_2]HOME_SEQUENCE = 1
```

One sequence, no synchronization

```
[JOINT_0]HOME_SEQUENCE = 0
[JOINT_1]HOME_SEQUENCE = 0
[JOINT_2]HOME_SEQUENCE = 0
```

One sequence, all joints synchronized

```
[JOINT_0]HOME_SEQUENCE = -1
[JOINT_1]HOME_SEQUENCE = -1
[JOINT_2]HOME_SEQUENCE = -1
```

9.3.6.12 VOLATILE_HOME

If this setting is true, this joint becomes unhomed whenever the machine transitions into the OFF state. This is appropriate for any joint that does not maintain position when the joint drive is off. Some stepper drives, especially microstep drives, may need this.

9.3.6.13 LOCKING_INDEXER

If this joint is a locking rotary indexer, it will unlock before homing, and lock afterward.

9.3.6.14 Immediate Homing

If a joint does not have home switches or does not have a logical home position like a rotary joint and you want that joint to home at the current position when the "Home All" button is pressed in the Axis gui, then the following INI entries for that joint are needed.

1. HOME_SEARCH_VEL = 0
2. HOME_LATCH_VEL = 0
3. HOME_USE_INDEX = NO
4. HOME equals to HOME_OFFSET
5. HOME_SEQUENCE = 0 (or other valid sequence number)

Note

The default values for unspecified HOME_SEARCH_VEL, HOME_LATCH_VEL, HOME_USE_INDEX, HOME, and HOME_OFFSET are **zero**, so they may be omitted when requesting immediate homing. A valid HOME_SEQUENCE number should usually be included since omitting a HOME_SEQUENCE eliminates the joint from **HOME ALL** behavior as noted above.

9.3.6.15 Inhibiting Homing

A HAL pin (motion.homing-inhibit) is provided to disallow homing initiation for both "Home All" and individual joint homing.

Some systems take advantage of the provisions for synchronizing final joint homing moves as controlled by negative [JOINT_N]HOME_SEQUENCE= INI file items. By default, the synchronization provisions disallow **joint** jogging prior to homing in order to prevent **joint** jogs that could misalign the machine (gantry racking for example).

System integrator can allow **joint** jogging prior to homing with HAL logic that switches the [JOINT_N]HOME items. This logic should also assert the **motion.homing-inhibit** pin to ensure that homing is not inadvertently initiated when **joint** jogging is enabled.

Example: Synced joints 0,1 using negative sequence (-1) for synchronized homing with a switch (allow_jjog) that selects a positive sequence (1) for individual **joint** jogging prior to homing (partial HAL code):

```
loadrt mux2          names=home_sequence_mux
loadrt conv_float_s32 names=home_sequence_s32
setp home_sequence_mux.in0 -1
setp home_sequence_mux.in1 1
addf home_sequence_mux servo-thread
addf home_sequence_s32 servo-thread
...
net home_seq_float <= home_sequence_mux.out
net home_seq_float => home_sequence_s32.in
net home_seq_s32   <= home_sequence_s32.out
net home_seq_s32   => ini.0.home_sequence
net home_seq_s32   => ini.1.home_sequence
...
# allow_jjog: pin created by a virtual panel or hardware switch
net hsequence_select <= allow_jjog
net hsequence_select => home_sequence_mux.sel
net hsequence_select => motion.homing-inhibit
```

Note

INI HAL pins (like ini.N.home_sequence) are not available until milltask starts so execution of the above HAL commands should be deferred using a postgui HAL file or a delayed [APPLICATION]APP=script.

Note

Realtime synchronization of joint jogging for multiple joints requires additional HAL connections for the Manual-Pulse-Generator (MPG) type jog pins (joint.N.enable, joint.N.scale, joint.N.counts).

An example simulation config (gantry_jjog.ini) that demonstrates joint jogging when using negative home sequences is located in the: configs/sim/axis/gantry/ directory.

9.4 I/O Control V2

9.4.1 Description

I/O control handles I/O tasks like coolant, toolchange, e-stop and lube. The signals are turned on and off in userspace with G-code or in the case of e-stop in HAL.

I/O Control V2 adds more toolchanger support for communication with the toolchanger.

- LinuxCNC originated abort and toolchanger fault: iocontrol reliably abort a change operation in progress (tool-change asserted). A toolchanger may at any time signal a fault which will abort the next M6. For instance, a toolchanger finding an empty pocket during a prepare operation should be able to signal a fault to iocontrol, and iocontrol act appropriately when the M6 change operation is executed.
- Communicate abort/fault cause : let iocontrol know why toolchanger caused an fault, and why iocontrol aborted. This is for UI purposes. It would be a candidate for a #5xxx parameter, and selective display in the UI.
- No race conditions between iocontrol and toolchanger: the protocol between iocontrol and toolchanger must be unambiguous with respect to which operation is signalled, and if a change operation is aborted or complete.
- Consistent view of state: Both parties must have a consistent view of the state at any point in time with respect to aborted versus completed, and tool number and pocket.
- Handshaked signaling an abort/fault: an abort signaled by LinuxCNC to the toolchanger, and a fault indicated from the toolchanger is handshaked to assure reliable signaling, and optionally force lock-step behaviour. Handshaking is optional and can be jumpered in HAL if not needed.
- Backwards compatibility: A toolchanger ignoring the iocontrol emc-abort line and sticking to old handling will "continue to work" (subject to race condition)

If you have strict timing requirements or simply need more I/O, consider using the realtime I/O provided by [motion](#) instead.

9.4.2 Usage

INI file options:

[EMCIO] section

PROTOCOL_VERSION = 2

Defaults to 2. Setting to 1 will emulate old iocontrol behaviour.

EMCIO = iov2 -support-start-change

You need to explicitly enable the start-change protocol by adding the -support-start-change option; otherwise the start-change pin remains low and start-change-ack is ignored. The reason for this is better backwards compatibility.

[TASK] section

IO_ERROR

Printf-style template for operator error display (negative toolchanger fault codes) . No quoting needed. Example: `IO_ERROR = Toolchanger fault %d`. Default: `toolchanger error %d`.

[EMC] section

DEBUG

To get a (quite detailed) trace, set either the RCS debugging flag (0x00000200) to turn on RCS debugging messages all over LinuxCNC or use the new iocontrol debugging bit (0x00001000) only for iov2 messages.

9.4.3 Pins

- `iocontrol.0.coolant-flood` (Bit, Out) TRUE when flood coolant is requested
- `iocontrol.0.coolant-mist` (Bit, Out) TRUE when mist coolant is requested
- `iocontrol.0.emc-enable-in` (Bit, In) Should be driven FALSE when an external estop condition exists.
- `iocontrol.0.lube` (Bit, Out) TRUE when lube is requested. This pin gets driven TRUE when the controller comes out of E-stop, and when the "Lube On" command gets sent to the controller. It gets driven FALSE when the controller goes into E-stop, and when the "Lube Off" command gets sent to the controller.
- `iocontrol.0.lube_level` (Bit, In) Should be driven FALSE when lubrication tank is empty.
- `iocontrol.0.tool-change` (Bit, Out) TRUE when a tool change is requested
- `iocontrol.0.tool-changed` (Bit, In) Should be driven TRUE when a tool change is completed.
- `iocontrol.0.tool-number` (s32, Out) Current tool number
- `iocontrol.0.tool-prep-number` (s32, Out) The number of the next tool, from the RS274NGC T-word
- `iocontrol.0.tool-prep-pocket` (s32, Out) This is the pocket number (location in the tool storage mechanism) of the tool requested by the most recent T-word.
- `iocontrol.0.tool-prepare` (Bit, Out) TRUE when a Tn tool prepare is requested
- `iocontrol.0.tool-prepared` (Bit, In) Should be driven TRUE when a tool prepare is completed.
- `iocontrol.0.user-enable-out` (Bit, Out) FALSE when an internal estop condition exists
- `iocontrol.0.user-request-enable` (Bit, Out) TRUE when the user has requested that estop be cleared

Additional pins added by I/O Control V2

- `emc-abort`: (Bit, Out) signals emc-originated abort to toolchanger.
- `emc-abort-ack`: (Bit, In) Acknowledge line from toolchanger for previous signal, or jumpered to `abort-tool-change` if not used in toolchanger. NB: after signaling an `emc-abort`, `iov2` will block until `emc-abort-ack` is raised.

- **emc-reason:** (S32, Out) convey cause for EMC-originated abort to toolchanger. Usage: UI informational. Valid during emc-abort TRUE.
- **start-change:** (Bit, Out) asserted at the very beginning of an M6 operation, before any spindle-off, quill-up, or move-to-toolchange-position operations are executed.
- **start-change-ack:** (Bit, In) acknowledgment line for start-change.
- **toolchanger-fault:** (Bit, In) toolchanger signals fault. This line is continuously monitored. A fault toggles a flag in iocontrol which is reflected in the toolchanger-faulted pin.
- **toolchanger-fault-ack:** (Bit, Out) handshake line for above signal. will be set by iov2 after above fault line TRUE is recognized and deasserted when toolchanger-fault drops. Toolchanger is free to interpret the ack; reading the -ack lines assures fault has been received and acted upon.
- **toolchanger-reason:** (S32, In) convey reason code for toolchanger-originated fault to iov2. Usage: signal whether to continue or abort the program, plus UI informational if negative. Read during toolchanger-fault TRUE. Non-zero values will cause an Axis operator operator message or error message, see below.
- **toolchanger-faulted:** (Bit, Out) signals toolchanger-notify line has toggled and toolchanger-reason-code was in the fault range. Next M6 will abort.
- **toolchanger-clear-fault:** (Bit, In) resets TC fault condition. Deasserts toolchanger-faulted if toolchanger-notify is line FALSE. Usage: UI - e.g. clear fault condition button.

9.4.4 Parameters

- **iocontrol.0.tool-prep-index** (s32, RO) IO's internal array index of the prepped tool requested by the most recent T-word. 0 if no tool is prepped. On random toolchanger machines this is tool's pocket number (ie, the same as the tool-prep-pocket pin), on non-random toolchanger machines this is a small integer corresponding to the tool's location in the internal representation of the tool table. This parameter returns to 0 after a successful tool change M6.

9.4.5 Communications

If LinuxCNC signals an abort for whatever reason, this is reflected in the emc-abort and emc-reason pins. The toolchanger is expected to acknowledge the emc-abort pin by raising the emc-abort-ack pin - iov2 will block until this is done. If you do not need the abort handshake feature, jumper them as follows:

```
net emc-abort-ack iocontrol.0.emc-abort iocontrol.0.emc-abort-ack
```

The emc-reason pin is considered valid during emc-abort being TRUE.

The reason codes are as follows for LinuxCNC internally generated aborts (see emc.hh ca line 321):

- `EMC_ABORT_TASK_EXEC_ERROR = 1,`
- `EMC_ABORT_AUX_ESTOP = 2,`
- `EMC_ABORT_MOTION_OR_IO_RCS_ERROR = 3,`
- `EMC_ABORT_TASK_STATE_OFF = 4,`
- `EMC_ABORT_TASK_STATE_ESTOP_RESET = 5,`
- `EMC_ABORT_TASK_STATE_ESTOP = 6,`
- `EMC_ABORT_TASK_STATE_NOT_ON = 7,`

- `EMC_ABORT_TASK_ABORT = 8`,
- `EMC_ABORT_USER = 100`

iov2 adds one more code, namely `EMC_ABORT_BY_TOOLCHANGER_FAULT = 101` which is signaled when an M6 aborts due to the toolchanger-faulted pin being TRUE.

To signal toolchanger faults to LinuxCNC, wire the toolchanger-fault pin, and optionally the toolchanger-reason and toolchanger-ack pins.

The toolchanger-fault triggers the fault condition, which is reflected in the toolchanger-faulted pin. This condition can be cleared by asserting the toolchanger-clear-fault pin, provided the toolchanger-fault pin is FALSE.

The value of the toolchanger-reason pin is used as follows:

- `toolchanger-reason > 0` : The toolchange process is not completed and the program continues, however parameter #5060 is set to 1.0 to indicate the fault. Parameter #5601 contains the value of the toolchanger-reason pin.
 - `toolchanger-reason = 0` : the program is aborted
 - `toolchanger-reason < 0` : the program is aborted and an operator error message is displayed by using the `IO_ERROR` template.

The usage of the toolchanger-fault-ack pin is optional. It will become TRUE when toolchanger-fault is raised and the toolchanger-reason pin has been read by iov2.

9.5 Lathe Configuration

9.5.1 Default Plane

When LinuxCNC's interpreter was first written, it was designed for mills. That is why the default plane is XY (G17). A normal lathe only uses the XZ plane (G18). To change the default plane place the following line in the .ini file in the RS274NGC section.

```
RS274NGC_STARTUP_CODE = G18
```

The above can be overwritten in a G-code program so always set important things in the preamble of the G-code file.

9.5.2 INI Settings

The following .ini settings are needed for lathe mode in Axis in addition to or replacing normal settings in the .ini file. These historical settings use identity kinematics (trivkins) and *three* joints (0,1,2) corresponding to coordinates x, y, z. The joint 1 for the unused y axis is required but not used in these historical configurations. Simulated lathe configs may use these historical settings. Gmoccapy also uses the mentioned settings, but does offer additional settings, check the [gmoccapy](#) Section for details.

```
[DISPLAY]
DISPLAY = axis
LATHE = 1
...

[KINS]
KINEMATICS = trivkins
```



```
JOINTS = 3

[TRAJ]
COORDINATES = X Z
...

[JOINT_0]
...
[JOINT_2]
...
[AXIS_X]
...
[AXIS_Z]
...
```

With joints_axes incorporation, a simpler configuration can be made with just the two required joints by specifying trivkins with the *coordinates=* parameter:

```
[DISPLAY]
DISPLAY = axis
LATHE = 1
...

[KINS]
KINEMATICS = trivkins coordinates=xz
JOINTS = 2

[TRAJ]
COORDINATES = X Z
...

[JOINT_0]
...
[JOINT_1]
...
[AXIS_X]
...
[AXIS_Z]
...
```

9.6 Remap Extending G-code

9.6.1 Introduction: Extending the RS274NGC Interpreter by Remapping Codes

9.6.1.1 A Definition: Remapping Codes

By *remapping codes* we mean one of the following:

1. define the semantics of new - that is, currently unallocated - M- or G-codes
2. redefine the semantics of a - currently limited - set of existing codes.

9.6.1.2 Why would you want to extend the RS274NGC Interpreter?

The set of codes (M,G,T,S,F) currently understood by the RS274NGC interpreter is fixed and cannot be extended by configuration options.

In particular, some of these codes implement a fixed sequence of steps to be executed. While some of these, like M6, can be moderately configured by activating or skipping some of these steps through ini file options, overall the behavior is fairly rigid. So - if you are happy with this situation, then this manual section is not for you.

In many cases, this means that supporting a non *out of the box* configuration or machine is either cumbersome or impossible, or requires resorting to changes at the C/C++ language level. The latter is unpopular for good reasons - changing internals requires in-depth understanding of interpreter internals, and moreover brings its own set of support issues. While it is conceivable that certain patches might make their way into the main LinuxCNC distribution, the result of this approach is a hodge-podge of special-case solutions.

A good example for this deficiency is tool change support in LinuxCNC: while random tool changers are well supported, it is next to impossible to reasonably define a configuration for a manual-tool change machine with, for example, an automatic tool length offset switch being visited after a tool change, and offsets set accordingly. Also, while a patch for a very specific rack tool changer exists, it has not found its way back into the main code base.

However, many of these things may be fixed by using an O-word procedure instead of a built in code - whenever the - insufficient - built in code is to be executed, call the O-word procedure instead. While possible, it is cumbersome - it requires source-editing of NGC programs, replacing all calls to the deficient code by an O-word procedure call.

In its simplest form a remapped code isn't much more than a spontaneous call to an O-word procedure. This happens behind the scenes - the procedure is visible at the configuration level, but not at the NGC program level.

Generally, the behavior of a remapped code may be defined in the following ways:

- you define a O-word subroutine which implements the desired behavior
- alternatively, you may employ a Python function which extends the interpreter's behavior.

How to glue things together M- and G-codes, and O-words subroutine calls have some fairly different syntax.

O-word procedures, for example, take positional parameters with a specific syntax like so:

```
o<test> call [1.234] [4.65]
```

whereas M- or G-codes typically take required or optional *word* parameters. For instance, G76 (threading) requires the P,Z,I,J and K words, and optionally takes the R,Q,H, E and L words.

So it isn't simply enough to say *whenever you encounter code X, please call procedure Y* - at least some checking and conversion of parameters needs to happen. This calls for some *glue code* between the new code, and its corresponding NGC procedure to execute before passing control to the NGC procedure.

This glue code is impossible to write as an O-word procedure itself since the RS274NGC language lacks the introspective capabilities and access into interpreter internal data structures to achieve the required effect. Doing the glue code in - again - C/C++ would be an inflexible and therefore unsatisfactory solution.

How Embedded Python fits in To make a simple situation easy and a complex situation solvable, the glue issue is addressed as follows:

- for simple situations, a built-in glue procedure (argspec) covers most common parameter passing requirements
- for remapping T,M6,M61,S,F there is some standard Python glue which should cover most situations, see [Standard Glue](#)
- for more complex situations, one can write your own Python glue to implement new behavior.

Embedded Python functions in the Interpreter started out as glue code, but turned out very useful well beyond that. Users familiar with Python will likely find it easier to write remapped codes, glue, O-word procedures etc in pure Python, without resorting to the somewhat cumbersome RS274NGC language at all.

A Word on Embedded Python Many people are familiar with *extending* the Python interpreter by C/C++ modules, and this is heavily used in LinuxCNC to access Task, HAL and Interpreter internals from Python scripts. *Extending Python* basically means: your Python script executes as *it is in the driver seat*, and may access non-Python code by importing and using extension modules written in C/C+++. Examples for this are the LinuxCNC `hal`, `gcode` and `emc` modules.

Embedded Python is a bit different and less commonly known: The main program is written in C/C++ and may use Python like a subroutine. This is powerful extension mechanism and the basis for the *scripting extensions* found in many successful software packages. Embedded Python code may access C/C++ variables and functions through a similar extension module method.

9.6.2 Getting started

Defining a code involves the following steps:

- pick a code - either use an unallocated code, or redefine an existing code
- deciding how parameters are handled
- decide if and how results are handled
- decide about the execution sequencing.

9.6.2.1 Builtin Remaps

Please note that currently only some existing codes can be redefined, while there are many *free* codes that may be available for remapping. When developing redefined existing code, it is a good idea to start with an unassigned G or M code, so that you can use both an existing behavior as well as a new one. When you're done, redefine the existing code to use your remapping configuration.

- The current set of unused M codes, available for user definition, can be found in the [unallocated M-codes section](#),
- For G-codes, see the [unallocated G-codes list](#).
- Existing codes that can be reassigned are listed in the [remappable codes section](#).

There are currently two complete python-only remaps that are available in `stdglue.py`.

- `ignore_m6`
- `index_lathe_tool_with_wear`

These are meant for use with lathe.

Lathes don't use M6 to index the tools, they use the T command.

This remap also adds wear offsets to the tool offset, ie. T201 would index to tool 2 (with tool 2's tool offset) and adds wear offset 1.

In the tool table, tools numbers above 10000 are wear offsets, ie in the tool table, tool 10001 would be wear offset 1.

Here is what you need in the INI to use them:

```
[RS274NGC]
REMAP=T python=index_lathe_tool_with_wear
REMAP=M6 python=ignore_m6

[PYTHON]
# where to find the Python code:

# code specific for this configuration
PATH_PREPEND=./

# generic support code - make sure this actually points to python-stdglue
PATH_APPEND=../../nc_files/remap_lib/python-stdglue/

# import the following Python module
TOPLEVEL=toplevel.py

# the higher the more verbose tracing of the Python plugin
LOG_LEVEL = 0
```

You must also add the required python file in your configuration folder.

Upgrade an existing configuration

9.6.2.2 Picking a code

Note that currently only a few existing codes may be redefined, whereas there are many *free* codes which might be made available by remapping. When developing a redefined existing code, it might be a good idea to start with an unallocated G- or M-code so both the existing and new behavior can be exercised. When done, redefine the existing code to use your remapping setup.

- the current set of unused M-codes open to user definition can be found [here](#),
- unallocated G-codes are listed [here](#).
- Existing codes which may be remapped are listed [here](#).

9.6.2.3 Parameter handling

Let's assume the new code will be defined by an NGC procedure, and needs some parameters, some of which might be required, others might be optional. We have the following options to feed values to the procedure:

1. extracting words from the current block and pass them to the procedure as parameters (like X22.34 or P47)
2. referring to [ini file variables](#)
3. referring to global variables (like #2200 = 47.11 or #<_global_param> = 315.2)

The first method is preferred for parameters of dynamic nature, , like positions. You need to define which words on the current block have any meaning for your new code, and specify how that is passed to the NGC procedure. Any easy way is to use the [argspec statement](#). A custom prolog might provide better error messages.

Using to ini file variables is most useful for referring to setup information for your machine, for instance a fixed position like a tool-length sensor position. The advantage of this method is that the parameters are fixed for your configuration regardless which NGC file you're currently executing.

Referring to global variables is always possible, but they are easily overlooked.

Note there's a limited supply of words which may be used as parameters, so one might need to fall back to the second and third methods if many parameters are needed.

9.6.2.4 Handling results

Your new code might succeed or fail, for instance if passed an invalid parameter combination. Or you might choose to *just execute* the procedure and disregard results, in which case there isn't much work to do.

Epilog handlers help in processing results of remap procedures - see the reference section.

9.6.2.5 Execution sequencing

Executable G-code words are classified into [modal groups](#), which also defines their relative execution behavior.

If a G-code block contains several executable words on a line, these words are executed in a predefined [order of execution](#), not in the order they appear in block.

When you define a new executable code, the interpreter does not yet know where your code fits into this scheme. For this reason, you need to choose an appropriate modal group for your code to execute in.

9.6.2.6 An minimal example remapped code

To give you an idea how the pieces fit together, let's explore a fairly minimal but complete remapped code definition. We choose an unallocated M-code and add the following option to the ini file:

```
[RS274NGC]
REMAP=M400 modalgroup=10 argspec=Pq ngc=myprocedure
```

In a nutshell, this means:

- The M400 code takes a required parameter P and an optional parameter Q. Other words in the current block are ignored with respect to the M400 code. If the P word is not present, fail execution with an error.
- when an M400 code is encountered, execute `myprocedure.ngc` along the other [modal group](#) 10 M-codes as per [order of execution](#).
- the value of P, and Q are available in the procedure as local named parameters. They may be referred to as `#<P>` and `#<Q>`. The procedure may test whether the Q word was present with the [EXISTS](#) built in function.

The file `myprocedure.ngc` is expected to exist in the `[DISPLAY]NC_FILES` or `[RS274NGC]SUBROUTINE_PATH` directory.

A detailed discussion of REMAP parameters is found in the reference section below.

9.6.3 Configuring Remapping

9.6.3.1 The REMAP statement

To remap a code, define it using the REMAP option in RS274NG section of your ini file. Use one REMAP line per remapped code.

The syntax of the REMAP is:

REMAP=<code> <options>

where <code> may be one of T,M6,M61,S,F (existing codes) or any of the unallocated [M-codes](#) or [G-codes](#).

It is an error to omit the <code> parameter.

The options of the REMAP statement are separated by whitespace. The options are keyword-value pairs and currently are:

modalgroup=<modal group>

G-codes

the only currently supported modal group is 1, which is also the default value if no group is given. Group 1 means *execute alongside other G-codes*.

M-codes

currently supported modal groups are: 5,6,7,8,9,10. If no modalgroup is give, it defaults to 10 (*execute after all other words in the block*).

T,S,F

for these the modal group is fixed and any modalgroup= option is ignored.

argspec=<argspec>

See [description of the argspec parameter options](#). Optional.

ngc=<ngc_basename>

Basename of an O-word subroutine file name. Do not specify an .ngc extension. Searched for in the directories specified in the directory specified in [DISPLAY]PROGRAM_PREFIX, then in [RS274NGC]SUBROUTINE_PATH. Mutually exclusive with python=. It is an error to omit both ngc= and python=.

python=<Python function name>

Instead of calling an ngc O-word procedure call a Python function. The function is expected to be defined in the module_basename.oword module. Mutually exclusive with ngc=.

prolog=<Python function name>

Before executing an ngc procedure, call this Python function. The function is expected to be defined in the module_basename.remap module. Optional.

epilog=<Python function name>

After executing an ngc procedure, call this Python function. The function is expected to be defined in the module_basename.remap module. Optional.

The python, prolog and epilog options require the Python Interpreter plugin to be [configured](#), and appropriate Python functions to be defined there so they can be referred to with these options.

The syntax for defining a new code, and redefining an existing code is identical.

9.6.3.2 Useful REMAP option combinations

Note that while many combinations of argspec options are possible, not all of them make sense. The following combinations are useful idioms:

argspec=<words> ngc=<procname> modalgroup=<group>

The recommended way to call an NGC procedure with a standard argspec parameter conversion. Used if argspec is good enough. Note it's not good enough for remapping the Tx and M6/M61 tool change codes.

prolog=<pythonprolog> ngc=<procname> epilog=<pythonepilog> modalgroup=<group>

Call a Python prolog function to take any preliminary steps, then call the NGC procedure. When done, call the Python epilog function to do any cleanup or result extraction work which cannot be handled in G-code. The most flexible way of remapping a code to an NGC procedure, since almost all of the Interpreter internal variables, and some internal functions may be accessed from the prolog and epilog handlers. Also, a longer rope to hang yourselves.

python=<pythonfunction> modalgroup=<group>

Directly call to a Python function without any argument conversion. The most powerful way of remapping a code and going straight to Python. Use this if you don't need an NGC procedure, or NGC is just getting in your way.

argspec=<words> python=<pythonfunction> modalgroup=<group>

Convert the argspec words and pass them to a Python function as keyword argument dictionary. Use it when you're too lazy to investigate words passed on the block yourself.

Note that if all you want to achieve is to call some Python code from G-code, there is the somewhat easier way of [calling Python functions like O-word procedures](#).

9.6.3.3 The argspec parameter

The argument specification (keyword argspec) describes required and optional words to be passed to an ngc procedure, as well as optional preconditions for that code to execute.

An argspec consists of 0 or more characters of the class [`@A-KMNP-Za-kmnp-z^>`] . It can be empty (like argspec=).

An empty argspec, or no argspec argument at all implies the remapped code does not receive any parameters from the block. It will ignore any extra parameters present.

Note that RS274NGC rules still apply - for instance you may use axis words (eg X,Y,Z) only in the context of a G-code.

Axis words may also only be used if the axis is enabled. If only XYZ are enabled, ABCUVW will not be available to be used in argspec.

Words FST will have the normal functions but will be available as variables in the remapped function. F will set feedrate, S will set spindle RPM, T will trigger the tool prepare function. Words FST should not be used if this behavior is not desired.

Words DEIJKPQR have no predefined function and are recommended for use as argspec parameters.

ABCDEFGHIJKPQRSTUVWXYZ

Defines a required word parameter: an uppercase letter specifies that the corresponding word **must** be present in the current block. The word's value will be passed as a local named parameter with a corresponding name. If the @ character is present in the argspec, it will be passed as positional parameter, see below.

abcdefghijklmnopqrstuvwxyz

Defines an optional word parameter: a lowercase letter specifies that the corresponding word **may** be present in the current block. If the word is present, the word's value will be passed as a local named parameter. If the @ character is present in the argspec, it will be passed as positional parameter, see below.

@

The @ (at-sign) tells argspec to pass words as positional parameters, in the order defined following the @ option. Note that when using positional parameter passing, a procedure cannot tell whether a word was present or not, see example below.

Tip

this helps with packaging existing NGC procedures as remapped codes. Existing procedures do expect positional parameters. With the @ option, you can avoid rewriting them to refer to local named parameters.

^

The ^ (caret) character specifies that the current spindle speed must be greater than zero (spindle running), otherwise the code fails with an appropriate error message.

>

The > (greater-than) character specifies that the current feed must be greater than zero, otherwise the code fails with an appropriate error message.

n

The n (greater-than) character specifies to pass the current line number in the `n` local named parameter.

By default, parameters are passed as local named parameter to an NGC procedure. These local parameters appear as *already set* when the procedure starts executing, which is different from existing semantics (local variables start out with value 0.0 and need to be explicitly assigned a value).

Optional word parameters may be tested for presence by the EXISTS(#<word>) idiom.

Example for named parameter passing to NGC procedures Assume the code is defined as

REMAP=M400 modalgroup=10 argspec=Pq ngc=m400

and m400.ngc looks as follows:

```
o<m400> sub
(P is required since it's uppercase in the argspec)
(debug, P word=#<P>)
(the q argspec is optional since its lowercase in the argspec. Use as follows:)
o100 if [EXISTS[#<q>]]
  (debug, Q word set: #<q>)
o100 endif
o<m400> endsub
M2
```

- executing M400 will fail with the message user-defined M400: missing: P
- executing M400 P123 will display P word=123.000000
- executing M400 P123 Q456 will display P word=123.000000 and Q word set: 456.000000

Example for positional parameter passing to NGC procedures Assume the code is defined as

REMAP=M410 modalgroup=10 argspec=@PQr ngc=m410

and m410.ngc looks as follows:

```
o<m410> sub
(debug, [1]=#1 [2]=#2 [3]=#3)
o<m410> endsub
M2
```

- executing M410 P10 will display m410.ngc: [1]=10.000000 [2]=0.000000
- executing M410 P10 Q20 will display m410.ngc: [1]=10.000000 [2]=20.000000

Note

you lose the capability to distinguish more than one optional parameter word, and you cannot tell whether an optional parameter was present but had the value 0, or was not present at all.

Simple example for named parameter passing to a Python function It's possible to define new codes *without* any NGC procedure. Here's a simple first example, a more complex one can be found in the next section.

Assume the code is defined as

REMAP=G88.6 modalgroup=1 argspec=XYZp python=g886

This instructs the interpreter to execute the Python function g886 in the module_basename.remap module which might look like so:

```
from interpreter import INTERP_OK
from emccanon import MESSAGE

def g886(self, **words):
    for key in words:
        MESSAGE("word '%s' = %f" % (key, words[key]))
    if words.has_key('p'):
        MESSAGE("the P word was present")
    MESSAGE("comment on this line: '%s'" % (self.blocks[self.remap_level].comment))
    return INTERP_OK
```

Try this with out with: g88.6 x1 y2 z3 g88.6 x1 y2 z3 p33 (a comment here)

You'll notice the gradual introduction of the embedded Python environment - see [here](#) for details. Note that with Python remapping functions, it make no sense to have Python prolog or epilog functions since it's executing a Python function in the first place.

Advanced example: Remapped codes in pure Python The interpreter and emccanon modules expose most of the Interpreter and some Canon internals, so many things which so far required coding in C/C++ can be now be done in Python.

The following example is based on the nc_files/involute.py script - but canned as a G-code with some parameter extraction and checking. It also demonstrates calling the interpreter recursively (see self.execute()).

Assuming a definition like so (NB: this does not use argspec):

REMAP=G88.1 modalgroup=1 py=involute

The involute function in python/remap.py listed below does all word extraction from the current block directly. Note that interpreter errors can be translated to Python exceptions. Remember this is *readahead time* - execution time errors cannot be trapped this way.

```

import sys
import traceback
from math import sin,cos

from interpreter import *
from emccanon import MESSAGE
from util import lineno, call_pydevd
# raises InterpreterException if execute() or read() fails
throw_exceptions = 1

def involute(self, **words):
    """ remap function with raw access to Interpreter internals """

    if self.debugmask & 0x20000000: call_pydevd() # USER2 debug flag

    if equal(self.feed_rate,0.0):
        return "feedrate > 0 required"

    if equal(self.speed,0.0):
        return "spindle speed > 0 required"

    plunge = 0.1 # if Z word was given, plunge - with reduced feed

    # inspect controlling block for relevant words
    c = self.blocks[self.remap_level]
    x0 = c.x_number if c.x_flag else 0
    y0 = c.y_number if c.y_flag else 0
    a = c.p_number if c.p_flag else 10
    old_z = self.current_z

    if self.debugmask & 0x10000000:
        print("x0=%f y0=%f a=%f old_z=%f" % (x0,y0,a,old_z))

    try:
        #self.execute("G3456") # would raise InterpreterException
        self.execute("G21",lineno())
        self.execute("G64 P0.001",lineno())
        self.execute("G0 X%f Y%f" % (x0,y0),lineno())

        if c.z_flag:
            feed = self.feed_rate
            self.execute("F%f G1 Z%f" % (feed * plunge, c.z_number),lineno())
            self.execute("F%f" % (feed),lineno())

        for i in range(100):
            t = i/10.
            x = x0 + a * (cos(t) + t * sin(t))
            y = y0 + a * (sin(t) - t * cos(t))
            self.execute("G1 X%f Y%f" % (x,y),lineno())

        if c.z_flag: # retract to starting height
            self.execute("G0 Z%f" % (old_z),lineno())

    except InterpreterException,e:
        msg = "%d: '%s' - %s" % (e.line_number,e.line_text, e.error_message)
    return msg

    return INTERP_OK

```

The examples described so far can be found in *configs/sim/axis/remap/getting-started* with complete working configurations.

9.6.4 Upgrading an existing configuration for remapping

The minimal prerequisites for using REMAP statements are as follows:

- the Python plug in must be activated by specifying a [PYTHON]TOPLEVEL=<path-to-toplevel-script> in the ini file.
- the toplevel script needs to import the remap module, which can be initially empty, but the import needs to be in place.
- The Python interpreter needs to find the remap.py module above, so the path to the directory where your Python modules live needs to be added with [PYTHON]PATH_APPEND=<path-to-your-local-Python-c>
- Recommended: import the stdglue handlers in the remap module. In this case Python also needs to find stdglue.py - we just copy it from the distribution so you can make local changes as needed. Depending on your installation the path to stdglue.py might vary.

Assuming your configuration lives under /home/user/xxx and the ini file is /home/user/xxx/xxx.ini, execute the following commands.

```
$ cd /home/user/xxx
$ mkdir python
$ cd python
$ cp /usr/share/linuxcnc/ncfiles/remap_lib/python-stdglue/stdglue.py .
$ echo 'from stdglue import *' >remap.py
$ echo 'import remap' >toplevel.py
```

Now edit /home/user/xxx/xxx.ini and add the following:

```
[PYTHON]
TOPLEVEL=/home/user/xxx/python/toplevel.py
PATH_APPEND=/home/user/xxx/python
```

Now verify that LinuxCNC comes up with no error messages - from a terminal window execute:

```
$ cd /home/user/xxx
$ linuxcnc xxx.ini
```

9.6.5 Remapping tool change-related codes: T, M6, M61

9.6.5.1 Overview

If you are unfamiliar with LinuxCNC internals, first read the [How tool change currently works](#) section (dire but necessary).

Note that when remapping an existing code, we completely disable [this codes' built in functionality](#) of the interpreter.

So our remapped code will need to do a bit more than just generating some commands to move the machine as we like - it will also need to replicate those steps from this sequence which are needed to keep the interpreter and task happy.

However, this does **not** affect the processing of tool change-related commands in task and iocontrol. This means when we execute [step 6b](#) this will still cause [iocontrol to do its thing](#).

Decisions, decisions:

- Do we want to use an O-word procedure or do it all in Python code?

- Is the iocontrol HAL sequence (tool-prepare/tool-prepared and tool-change/tool-changed pins) good enough or do we need a different kind of HAL interaction for our tool changer (for example: more HAL pins involved with a different interaction sequence)?

Depending on the answer, we have four different scenarios:

- When using an O-word procedure, we need prolog and epilog functions
- if using all Python code and no O-word procedure, a Python function is enough
- when using the iocontrol pins, our O-word procedure or Python code will contain mostly moves
- when we need a more complex interaction than offered by iocontrol, we need to completely define our own interaction, using `motion.digital*` and `motion.analog*` pins, and essentially ignore the iocontrol pins by looping them.

Note

If you hate O-word procedures and love Python, you're free to do it all in Python, in which case you would just have a `python=<function>` spec in the REMAP statement. But assuming most folks would be interested in using O-word procedures because they are more familiar with that, we'll do that as the first example.

So the overall approach for our first example will be:

1. we'd like to do as much as possible with G-code in an O-word procedure for flexibility. That includes all HAL interaction which would normally be handled by iocontrol - because we rather would want to do clever things with moves, probes, HAL pin I/O and so forth.
2. we'll try to minimize Python code to the extent needed to keep the interpreter happy, and cause task to actually do anything. That will go into the prolog and epilog Python functions.

9.6.5.2 Understanding the role of iocontrol with remapped tool change codes

Iocontrol provides two HAL interaction sequences we might or might not use:

- when the NML message queued by a `SELECT_TOOL()` canon command is executed, this triggers the "raise tool-prepare and wait for tool-prepared to become high" HAL sequence in iocontrol, besides setting the XXXX pins
- when the NML message queued by the `CHANGE_TOOL()` canon command is executed, this triggers the "raise tool-change and wait for tool-changed to become high" HAL sequence in iocontrol, besides setting the XXXX pins

What you need to decide is whether the existing iocontrol HAL sequences are sufficient to drive your changer. Maybe you need a different interaction sequence - for instance more HAL pins, or maybe a more complex interaction. Depending on the answer, we might continue to use the existing iocontrol HAL sequences, or define our own ones.

For the sake of documentation, we'll disable these iocontrol sequences, and roll our own - the result will look and feel like the existing interaction, but now we have complete control over them because they are executed in our own O-word procedure.

So what we'll do is use some `motion.digital-*` and `motion.analog-*` pins, and the associated M62 .. M68 commands to do our own HAL interaction in our O-word procedure, and those will effectively replace the iocontrol *tool-prepare/tool-prepared* and *tool-change/tool-changed* sequences. So we'll

define our pins replacing existing iocontrol pins functionally, and go ahead and make the iocontrol interactions a loop. We'll use the following correspondence in our example:

Iocontrol pin correspondence in the examples

iocontrol.0 pin	motion pin
tool-prepare	digital-out-00
tool-prepared	digital-in-00
tool-change	digital-out-01
tool-changed	digital-in-01
tool-prep-number	analog-out-00
tool-prep-pocket	analog-out-01
tool-number	analog-out-02

Let us assume you want to redefine the M6 command, and replace it by an O-word procedure, but other than that things *should continue to work*.

So what our O-word procedure would do is to replace the steps [outlined here](#). Looking through these steps you'll find that NGC code can be used for most of them, but not all. So the stuff NGC can't handle will be done in Python prolog and epilog functions.

9.6.5.3 Specifying the M6 replacement

To convey the idea, we just replace the built in M6 semantics with our own. Once that works, you may go ahead and place any actions you see fit into the O-word procedure.

Going through the [steps](#), we find:

1. check for T command already executed - **execute in Python prolog**
2. check for cutter compensation being active - **execute in Python prolog**
3. stop the spindle if needed - **can be done in NGC**
4. quill up - **can be done in NGC**
5. if TOOL_CHANGE_AT_G30 was set:
 - a. move the A, B and C indexers if applicable - **can be done in NGC**
 - b. generate rapid move to the G30 position - **can be done in NGC**
6. send a CHANGE_TOOL Canon command to task - **execute in Python epilog**
7. set the numberer parameters 5400-5413 according to the new tool - **execute in Python epilog**
8. signal to task to stop calling the interpreter for readahead until tool change complete - **execute in Python epilog**

So we need a prolog, and an epilog. Lets assume our ini file incantation of the M6 remap looks as follows:

```
REMAP=M6 modalgroup=6 prolog=change_prolog ngc=change epilog=change_epilog
```

So the prolog covering steps 1 and 2 would look like so - we decide to pass a few variables to the remap procedure which can be inspected and changed there, or used in a message. Those are: tool_in_spindle, selected_tool (tool numbers) and their respective tooldata indices current_pocket and selected_pocket:

Note

The legacy names **selected_pocket** and **current_pocket** actually reference a sequential tooldata index for tool items loaded from a tool table ([EMCIO]TOOL_TABLE) or via a tooldata database ([EMCIO]DB_PROGRAM)

```
def change_prolog(self, **words):
    try:
        if self.selected_pocket < 0:
            return "M6: no tool prepared"

        if self.cutter_comp_side:
            return "Cannot change tools with cutter radius compensation on"

        self.params["tool_in_spindle"] = self.current_tool
        self.params["selected_tool"] = self.selected_tool
        self.params["current_pocket"] = self.current_pocket
        self.params["selected_pocket"] = self.selected_pocket
        return INTERP_OK
    except Exception, e:
        return "M6/change_prolog: %s" % (e)
```

You will find that most prolog functions look very similar: first test that all preconditions for executing the code hold, then prepare the environment - inject variables and/or do any preparatory processing steps which cannot easily be done in NGC code; then hand off to the NGC procedure by returning INTERP_OK.

Our first iteration of the O-word procedure is unexciting - just verify we got parameters right, and signal success by returning a positive value; steps 3-5 would eventually be covered here (see [here](#) for the variables referring to ini file settings):

```
0<change> sub
(debug, change: current_tool=#<current_tool>)
(debug, change: selected_pocket=#<selected_pocket>)
;
; insert any G-code which you see fit here, eg:
; G0 #<_ini[setup]tc_x> #<_ini[setup]tc_y> #<_ini[setup]tc_z>
;
0<change> endsub [1]
m2
```

Assuming success of change.ngc, we need to mop up steps 6-8:

```
def change_epilog(self, **words):
    try:
        if self.return_value > 0.0:
            # commit change
            self.selected_pocket = int(self.params["selected_pocket"])
            emccanon.CHANGE_TOOL(self.selected_pocket)
            # cause a sync()
            self.tool_change_flag = True
            self.set_tool_parameters()
            return INTERP_OK
        else:
            return "M6 aborted (return code %.1f)" % (self.return_value)

    except Exception, e:
        return "M6/change_epilog: %s" % (e)
```

This replacement M6 is compatible with the built in code, except steps 3-5 need to be filled in with your NGC code.

Again, most epilogs have a common scheme: first, determine whether things went right in the remap procedure, then do any commit and cleanup actions which can't be done in NGC code.

9.6.5.4 Configuring iocontrol with a remapped M6

Note that the sequence of operations has changed: we do everything required in the O-word procedure - including any HAL pin setting/reading to get a changer going, and to acknowledge a tool change - likely with `motion.digital-*` and `motion-analog-*` IO pins. When we finally execute the `CHANGE_TOOL()` command, all movements and HAL interactions are already completed.

Normally only now iocontrol would do its thing as outlined [here](#). However, we don't need the HAL pin wiggling anymore - all iocontrol is left to do is to accept we're done with prepare and change.

This means that the corresponding iocontrol pins have no function any more. Therefore, we configure iocontrol to immediately acknowledge a change by configuring like so:

```
# loop change signals when remapping M6
net tool-change-loop iocontrol.0.tool-change iocontrol.0.tool-changed
```

If you for some reason want to remap Tx (prepare), the corresponding iocontrol pins need to be looped as well.

9.6.5.5 Writing the change and prepare O-word procedures

The standard prologs and epilogs found in `ncfiles/remap_lib/python-stdglue/stdglue.py` pass a few *exposed parameters* to the remap procedure.

An *exposed parameter* is a named local variable visible in a remap procedure which corresponds to interpreter-internal variable which is relevant for the current remap. Exposed parameters are set up in the respective prolog, and inspected in the epilog. They can be changed in the remap procedure and the change will be picked up in the epilog. The exposed parameters for remappable built in codes are:

- T (prepare_prolog): #<tool> , #<pocket>
- M6 (change_prolog): #<tool_in_spindle>, #<selected_tool>, #<current_pocket>, #<selected_pocket>
- M61 (settool_prolog): #<tool> , #<pocket>
- S (setspeed_prolog): #<speed>
- F (setfeed_prolog): #<feed>

If you have specific needs for extra parameters to be made visible, that can simply be added to the prolog - practically all of the interpreter internals are visible to Python.

9.6.5.6 Making minimal changes to the built in codes, including M6

Remember that normally remapping a code completely disables all internal processing for that code. However, in some situations it might be sufficient to add a few codes around the existing M6 built in implementation, like a tool length probe, but other than that retain the behavior of the built in M6.

Since this might be a common scenario, the built in behavior of remapped codes has been made available within the remap procedure. The interpreter detects that you are referring to a remapped code within the procedure which is supposed to redefine its behavior. In this case, the built in behavior is used - this currently is enabled for the set: M6, M61, T, S, F. Note that otherwise referring to a code within its own remap procedure would be a error - a remapping recursion.

Slightly twisting a built in would look like so (in the case of M6):

```
REMAP=M6   modalgroup=6   ngc=mychange
```

```
o<mychange> sub
M6 (use built in M6 behavior)
(.. move to tool length switch, probe and set tool length..)
o<mychange> endsub
m2
```



Caution

when redefining a built in code, **do not specify any leading zeroes in G- or M-codes** - for example, say REMAP=M1 .., not REMAP=M01

See the configs/sim/axis/remap/extend-builtins directory for a complete configuration which is the recommended starting point for own work when extending built in codes.

9.6.5.7 Specifying the T (prepare) replacement

If you're confident with the [default implementation](#), you wouldn't need to do this. But remapping is also a way to work around deficiencies in the current implementation, for instance to not block until the "tool-prepared" pin is set.

What you could do, for instance, is: - in a remapped T, just set the equivalent of the "tool-prepare" pin, but **not** wait for "tool-prepared" here - in the corresponding remapped M6, wait for the "tool-prepared" at the very beginning of the O-word procedure.

Again, the iocontrol tool-prepare/tool-prepared pins would be unused and replaced by motion.* pins, so those would pins must be looped:

```
# loop prepare signals when remapping T
net tool-prep-loop iocontrol.0.tool-prepare iocontrol.0.tool-prepared
```

So, here's the setup for a remapped T:

```
REMAP=T   prolog=prepare_prolog   epilog=prepare_epilog   ngc=prepare
```

```
def prepare_prolog(self,**words):
    try:
        cblock = self.blocks[self.remap_level]
        if not cblock.t_flag:
            return "T requires a tool number"

        tool = cblock.t_number
        if tool:
            (status, pocket) = self.find_tool_pocket(tool)
            if status != INTERP_OK:
                return "T%d: pocket not found" % (tool)
        else:
            pocket = -1 # this is a T0 - tool unload

        # these variables will be visible in the ngc oword sub
        # as #<tool> and #<pocket> local variables, and can be
        # modified there - the epilog will retrieve the changed
        # values
        self.params["tool"] = tool
        self.params["pocket"] = pocket
```



```

        return INTERP_OK
    except Exception, e:
        return "T%d/prepare_prolog: %s" % (int(words['t']), e)

```

The minimal ngc prepare procedure again looks like so:

```

o<prepare> sub
; returning a positive value to commit:
o<prepare> endsub [1]
m2

```

And the epillog:

```

def prepare_epilog(self, **words):
    try:
        if self.return_value > 0:
            self.selected_tool = int(self.params["tool"])
            self.selected_pocket = int(self.params["pocket"])
            emccanon.SELECT_TOOL(self.selected_tool)
            return INTERP_OK
        else:
            return "T%d: aborted (return code %.1f)" % (int(self.params["tool"]),self.return_value)
    except Exception, e:
        return "T%d/prepare_epilog: %s" % (tool,e)

```

prepare_prolog and prepare_epilog are part of the *standard glue* provided by `nc_files/remap_lib/python-stdglue/stdglue.py`. This module is intended to cover most standard remapping situations in a common way.

9.6.5.8 Error handling: dealing with abort

The built in tool change procedure has some precautions for dealing with a program abort (e.g. hitting Escape in Axis during a change). Your remapped function has none of this, therefore some explicit cleanup might be needed if a remapped code is aborted. In particular, a remap procedure might establish modal settings which are undesirable to have active after an abort. For instance, if your remap procedure has motion codes (G0,G1,G38..) and the remap is aborted, then the last modal code will remain active. However, you very likely want to have any modal motion canceled when the remap is aborted.

The way to do this is by using the [RS274NGC]ON_ABORT_COMMAND feature. This ini option specifies a O-word procedure call which is executed if task for some reason aborts program execution. `on_abort` receives a single parameter indicating the cause for calling the abort procedure, which might be used for conditional cleanup.

The reasons are defined in `nml_intf/emc.hh`

```

EMC_ABORT_TASK_EXEC_ERROR = 1,
EMC_ABORT_AUX_ESTOP = 2,
EMC_ABORT_MOTION_OR_IO_RCS_ERROR = 3,
EMC_ABORT_TASK_STATE_OFF = 4,
EMC_ABORT_TASK_STATE_ESTOP_RESET = 5,
EMC_ABORT_TASK_STATE_ESTOP = 6,
EMC_ABORT_TASK_STATE_NOT_ON = 7,
EMC_ABORT_TASK_ABORT = 8,
EMC_ABORT_INTERPRETER_ERROR = 9,           // interpreter failed during readahead
EMC_ABORT_INTERPRETER_ERROR_MDI = 10,     // interpreter failed during MDI execution
EMC_ABORT_USER = 100 // user-defined abort codes start here

```

```
[RS274NGC]
ON_ABORT_COMMAND=0 <on_abort> call
```

The suggested `on_abort` procedure would look like so (adapt to your needs):

```
o<on_abort> sub

G54 (origin offsets are set to the default)
G17 (select XY plane)
G90 (absolute)
G94 (feed mode: units/minute)
M48 (set feed and speed overrides)
G40 (cutter compensation off)
M5 (spindle off)
G80 (cancel modal motion)
M9 (mist and coolant off)

o100 if [#1 eq 5]
    (machine on)
o100 elseif [#1 eq 6]
    (machine off)
o100 elseif [#1 eq 7]
    (estopped)
o100 elseif [#1 eq 8]
    (msg, abort pressed)
o100 else
    (DEBUG, error parameter is [#1])
o100 endif

o<on_abort> endsub
m2
```



Caution

Never use an M2 in a O-word subroutine, including this one. It will cause hard-to-find errors. For instance, using an M2 in a subroutine will not end the subroutine properly and will leave the subroutine NGC file open, not your main program.

Make sure `on_abort.ngc` is along the interpreter search path (recommended location: `SUBROUTINE_PATH` so as not to clutter your `NC_FILES` directory with internal procedures).

Statements in that procedure typically would assure that post-abort any state has been cleaned up, like HAL pins properly reset. For an example, see `configs/sim/axis/remap/rack-toolchange`.

Note that terminating a remapped code by returning `INTERP_ERROR` from the epilog (see previous section) will also cause the `on_abort` procedure to be called.

9.6.5.9 Error handling: failing a remapped code NGC procedure

If you determine in your handler procedure that some error condition occurred, do not use M2 to end your handler - see above:

If displaying an operator error message and stopping the current program is good enough, use the `(abort, <message>)` feature to terminate the handler with an error message. Note that you can substitute numbered, named, ini and HAL parameters in the text like in this example (see also `tests/interp/ab`

```
o100 if [...] (some error condition)
    (abort, Bad Things! p42=#42 q=#<q> ini=#<_ini[a]> pin=#<_hal[component.pin])
o100 endif
```

Note

ini and HAL variable expansion is optional and can be disabled in the [INI file](#)

If more fine grained recovery action is needed, use the idiom laid out in the previous example:

- define an epilog function, even if it's just to signal an error condition
- pass a negative value from the handler to signal the error
- inspect the return value in the epilog function.
- take any recovery action needed
- return the error message string from the handler, which will set the interpreter error message and abort the program (pretty much like `abort, message=`)

This error message will be displayed in the UI, and returning `INTERP_ERROR` will cause this error handled like any other runtime error.

Note that both `(abort, msg)` and returning `INTERP_ERROR` from an epilog will cause any `ON_ABORT` handler to be called as well if defined (see previous section).

9.6.6 Remapping other existing codes: S, M0, M1, M60

9.6.6.1 Automatic gear selection be remapping S (set spindle speed)

A potential use for a remapped S code would be *automatic gear selection* depending on speed. In the remap procedure one would test for the desired speed attainable given the current gear setting, and change gears appropriately if not.

9.6.6.2 Adjusting the behavior of M0, M1, M60

A use case for remapping M0/M1 would be to customize the behavior of the existing code. For instance, it could be desirable to turn off the spindle, mist and flood during an M0 or M1 program pause, and turn these settings back on when the program is resumed.

For a complete example doing just that, see *configs/sim/axis/remap/extend-builtins/*, which adapts M1 as laid out above.

9.6.7 Creating new G-code cycles

A G-code cycle as used here is meant to behave as follows:

- On first invocation, the associated words are collected and the G-code cycle is executed.
- If subsequent lines just continue parameter words applicable to this code, but no new G-code, the previous G-code is re-executed with the parameters changed accordingly.

An example: Assume you have G84.3 defined as remapped G-code cycle with the following ini segment (see [here](#) for a detailed description of `cycle_prolog` and `cycle_epilog`):

```
[RS274NGC]
# A cycle with an oword procedure: G84.3 <X- Y- Z- Q- P->
REMAP=G84.3 argspec=xyzabcuvwpr prolog=cycle_prolog ngc=g843 epilog=cycle_epilog modalgroup ←
=1
```

Executing the following lines:

```
g17
(1)  g84.3 x1 y2 z3  r1
(2)  x3 y4 p2
(3)  x6 y7 z5
(4)  G80
```

causes the following (note *R* is sticky, and *Z* is sticky since the plane is *XY*):

1. g843.ngc is called with words x=1, y=2, z=3, r=1
2. g843.ngc is called with words x=3, y=4, z=3, p=2, r=1
3. g843.ngc is called with words x=6, y=7, z=3, r=1
4. The G84.3 cycle is canceled.

Besides creating new cycles, this provides an easy method for repackaging existing G-codes which do not behave as cycles. For instance, the G33.1 Rigid Tapping code does not behave as a cycle. With such a wrapper, a new code can be easily created which uses G33.1 but behaves as a cycle.

See *configs/sim/axis/remap/cycle* for a complete example of this feature. It contains two cycles, one with an NGC procedure like above, and a cycle example using just Python.

9.6.8 Configuring Embedded Python

The Python plugin serves both the interpreter, and task if so configured, and hence has its own section PYTHON in the ini file.

9.6.8.1 Python plugin : ini file configuration

[PYTHON]

TOPLEVEL = <filename>

filename of the initial Python script to execute on startup. This script is responsible for setting up the package name structure, see below.

PATH_PREPEND = <directory>

prepend this directory to PYTHON_PATH. A repeating group.

PATH_APPEND =<directory>

append this directory to PYTHON_PATH. A repeating group.

LOG_LEVEL = <integer>

log level of plugin-related actions. Increase this if you suspect problems. Can be very verbose.

RELOAD_ON_CHANGE = [0|1]

reload the *TOPLEVEL* script if the file was changed. Handy for debugging but currently incurs some runtime overhead. Turn this off for production configurations.

PYTHON_TASK = [0|1]

Start the Python task plug in. Experimental. See xxx.

9.6.8.2 Executing Python statements from the interpreter

For ad-hoc execution of commands the Python *hot comment* has been added. Python output by default goes to stdout, so you need to start LinuxCNC from a terminal window to see results. Example (eg. in the MDI window):

```
;py,print(2*3)
```

Note that the interpreter instance is available here as `self`, so you could also run:

```
;py,print(self.tool_table[0].toolno)
```

The `emcStatus` structure is accessible, too:

```
;py,from emctask import *
;py,print(emcstat.io.aux.estop)
```

9.6.9 Programming Embedded Python in the RS274NGC Interpreter

9.6.9.1 The Python plugin namespace

The namespace is expected to be laid out as follows:

oword

Any callables in this module are candidates for Python O-word procedures. Note that the Python oword module is checked **before** testing for a NGC procedure with the same name - in effect names in oword will hide NGC files of the same basename.

remap

Python callables referenced in an argspec prolog,epilog or python option are expected to be found here.

namedparams

Python functions in this module extend or redefine the namespace of predefined named parameters, see [adding predefined parameters](#).

task

Task-related callables are expected here.

9.6.9.2 The Interpreter as seen from Python

The interpreter is an existing C++ class (*Interp*) defined in `src/emc/rs274ngc`. Conceptually all `oword.<function>` and `remap.<function>` Python calls are methods of this *Interp* class, although there is no explicit Python definition of this class (it's a *Boost.Python* wrapper instance) and hence receive the as the first parameter `self` which can be used to access internals.

9.6.9.3 The Interpreter `__init__` and `__delete__` functions

If the `TOPLEVEL` module defines a function `__init__`, it will be called once the interpreter is fully configured (ini file read, and state synchronized with the world model).

If the `TOPLEVEL` module defines a function `__delete__`, it will be called once before the interpreter is shutdown and after the persistent parameters have been saved to the `PARAMETER_FILE`.

Note at this time, the `__delete__` handler does not work for interpreter instances created by importing the `gcode` module. If you need an equivalent functionality there (which is quite unlikely), please consider the Python `atexit` module.

```
# this would be defined in the TOPLEVEL module

def __init__(self):
    # add any one-time initialization here
    if self.task:
        # this is the milltask instance of interp
        pass
    else:
        # this is a non-milltask instance of interp
        pass

def __delete__(self):
    # add any cleanup/state saving actions here
    if self.task: # as above
        pass
    else:
        pass
```

This function may be used to initialize any Python-side attributes which might be needed later, for instance in remap or oword functions, and save or restore state beyond what `PARAMETER_FILE` provides.

If there are setup or cleanup actions which are to happen only in the milltask Interpreter instance (as opposed to the interpreter instance which sits in the gcode Python module and serves preview/progress display purposes but nothing else), this can be tested for by [evaluating `self.task`](#).

An example use of `__init__` and `__delete__` can be found in `configs/sim/axis/remap/cycle/python/toplevel.py` initialising attributes needed to handle cycles in `ncfiles/remap_lib/python-stdglue/stdglue.py` (and imported into `configs/sim/axis/remap/cycle/python/remap.py`).

9.6.9.4 Calling conventions: NGC to Python

Python code is called from NGC in the following situations:

- during normal program execution:
 - when an O-word call like `O<proc> call` is executed and the name `oword.proc` is defined and callable
 - when a comment like `;py,<Python statement>` is executed
- during execution of a remapped code: any `prolog=`, `python=` and `epilog=` handlers.

Calling O-word Python subroutines

Arguments:

self

the interpreter instance

***args**

the list of actual positional parameters. Since the number of actual parameters may vary, it is best to use this style of declaration:

```
# this would be defined in the oword module
def mysub(self, *args):
    print("number of parameters passed:", len(args))
    for a in args:
        print(a)
```

Return values of O-word Python subroutines Just as NGC procedures may return values, so do O-word Python subroutines. They are expected to either:

- return no value (no return statement or the value None)
- a float or int value
- a string, this means *this is an error message, abort the program*. Works like (abort, msg).

Any other return value type will raise a Python exception.

In a calling NGC environment, the following predefined named parameters are available:

#<_value>

value returned by the last procedure called. Initialized to 0.0 on startup. Exposed in Interp as self.return_value (float).

#<_value_returned>

indicates the last procedure called did return or ends with an explicit value. 1.0 if true. Set to 0.0 on each call. Exposed in Interp as self.value_returned (int).

See also tests/interp/value_returned for an example.

Calling conventions for prolog= and epilog= subroutines Arguments are:

self

the interpreter instance

words

keyword parameter dictionary. If an argspec was present, words are collected from the current block accordingly and passed in the dictionary for convenience (the words could as well be retrieved directly from the calling block, but this requires more knowledge of interpreter internals). If no argspec was passed, or only optional values were specified and none of these was present in the calling block, this dict is empty. Word names are converted to lowercase.

Example call:

```
def minimal_prolog(self, **words): # in remap module
    print(len(words), " words passed")
    for w in words:
        print("%s: %s" % (w, words[w]))
    if words['p'] < 78: # NB: could raise an exception if p were optional
        return "failing miserably"
    return INTERP_OK
```

Return values:

INTERP_OK

return this on success. You need to import this from interpreter.

"a message text"

returning a string from a handler means *this is an error message, abort the program*. Works like (abort, msg).

Calling conventions for python= subroutines Arguments are:

self

the interpreter instance

words

keyword parameter dictionary. the same kwargs dictionary as prologs and epilogs (see above).

The minimum python= function example:

```
def useless(self, **words): # in remap module
    return INTERP_OK
```

Return values:

INTERP_OK

return this on success

"a message text"

returning a string from a handler means *this is an error message, abort the program*. Works like (abort, msg).

If the handler needs to execute a *queuebuster* operation (tool change, probe, HAL pin reading) it is supposed to suspend execution with the following statement:

yield INTERP_EXECUTE_FINISH

This signals task to stop read ahead, execute all queued operations, execute the *queue-buster* operation, synchronize interpreter state with machine state, and then signal the interpreter to continue. At this point the function is resumed at the statement following the `yield ..` statement.

Dealing with queue-buster: Probe, Tool change and waiting for a HAL pin Queue busters interrupt a procedure at the point where such an operation is called, hence the procedure needs to be restarted after the interpreter `synch()`. When this happens the procedure needs to know if it is restarted, and where to continue. The Python generator method is used to deal with procedure restart.

This demonstrates call continuation with a single point-of-restart:

```
def read_pin(self,*args):
    # wait 5secs for digital-input 00 to go high
    emccanon.WAIT(0,1,2,5.0)
    # cede control after executing the queue buster:
    yield INTERP_EXECUTE_FINISH
    # post-sync() execution resumes here:
    pin_status = emccanon.GET_EXTERNAL_DIGITAL_INPUT(0,0);
    print("pin status=",pin_status)
```

**Warning**

The *yield* feature is fragile. The following restrictions apply to the usage of *yield INTERP_EXECUTE_FINISH*:

- Python code executing a *yield INTERP_EXECUTE_FINISH* must be part of a remap procedure. Yield does not work in a Python oword procedure.
- A Python remap subroutine containing *yield INTERP_EXECUTE_FINISH* statement may not return a value, as with normal Python yield statements.
- Code following a yield may not recursively call the interpreter, like with `self.execute("<mdi command>")`. This is an architectural restriction of the interpreter and is not fixable without a major redesign.

9.6.9.5 Calling conventions: Python to NGC

NGC code is executed from Python when:

- the method `self.execute(<NGC code>[,<line number>])` is executed
- during execution of a remapped code, if a `prolog=` function is defined, the NGC procedure given in `ngc=` is executed immediately thereafter.

The prolog handler does not call the handler, but it prepares its call environment, for instance by setting up predefined local parameters.

Inserting parameters in a prolog, and retrieving them in an epilog Conceptually a prolog and an epilog execute at the same call level like the O-word procedure, that is: after the subroutine call is set up, and before the subroutine endsub or return.

This means that any local variable created in a prolog will be a local variable in the O-word procedure, and any local variables created in the O-word procedure are still accessible when the epilog executes.

The `self.params` array handles reading and setting numbered and named parameters. If a named parameter begins with `_` (underscore), it is assumed to be a global parameter; if not, it is local to the calling procedure. Also, numbered parameters in the range 1..30 are treated like local variables; their original values are restored on return/endsub from an O-word procedure.

Here is an example remapped code demonstrating insertion and extraction of parameters into/from the O-word procedure:

```
REMAP=m300 prolog=insert_param ngc=testparam epilog=retrieve_param modalgroup=10
```

```
def insert_param(self, **words): # in the remap module
    print("insert_param call level=",self.call_level)
    self.params["myname"] = 123
    self.params[1] = 345
    self.params[2] = 678
    return INTERP_OK

def retrieve_param(self, **words):
    print("retrieve_param call level=",self.call_level)
    print("#1=", self.params[1])
    print("#2=", self.params[2])
    try:
        print("result=", self.params["result"])
    except Exception,e:
        return "testparam forgot to assign #<result>"
    return INTERP_OK
```

```
o<testparam> sub
(debug, call_level=#<_call_level> myname=#<myname>)
; try commenting out the next line and run again
#<result> = [#<myname> * 3]
#1 = [#1 * 5]
#2 = [#2 * 3]
o<testparam> endsub
m2
```

`self.params()` returns a list of all variable names currently defined. Since `myname` is local, it goes away after the epilog finishes.

Calling the interpreter from Python You can recursively call the interpreter from Python code as follows:

```
self.execute(<NGC code>[,<line number>])
```

Examples:

```
self.execute("G1 X%f Y%f" % (x,y))
self.execute("O <myprocedure> call", currentline)
```

You might want to test for the return value being < INTERP_MIN_ERROR. If you're using lots of execute() statements, it's probably easier to trap InterpreterException as per below.



Caution

The parameter insertion/retrieval method described in the previous section does not work in this case. It is good enough for just executing simple NGC commands or a procedure call and advanced introspection into the procedure, and passing of local named parameters is not needed. The recursive call feature is fragile.

Interpreter Exception during execute() if interpreter.throw_exceptions is nonzero (default 1), and self.execute() returns an error, the exception InterpreterException is raised. InterpreterException has the following attributes:

line_number

where the error occurred

line_text

the NGC statement causing the error

error_message

the interpreter's error message

Errors can be trapped in the following Pythonic way:

```
import interpreter
interpreter.throw_exceptions = 1
...
try:
    self.execute("G3456") # raise InterpreterException
except InterpreterException,e:
    msg = "%d: '%s' - %s" % (e.line_number,e.line_text, e.error_message)
    return msg # replace builtin error message
```

Canon The canon layer is practically all free functions. Example:

```
import emccanon
def example(self,*args):
    ....
    emccanon.STRAIGHT_TRAVERSE(line,x0,y0,z0,0,0,0,0,0,0)
    emccanon.STRAIGHT_FEED(line,x1,y1,z1,0,0,0,0,0,0)
    ...
    return INTERP_OK
```

The actual canon functions are declared in src/emc/nml_intf/canon.hh and implemented in src/emc/task. The implementation of the Python functions can be found in src/emc/rs274ncg/canonmodule.cc.

9.6.9.6 Built in modules

The following modules are built in:

interpreter

exposes internals of the Interp class. See `src/emc/rs274ngc/interpmodule.cc`, and the `tests/remap/regression` test.

emccanon

exposes most calls of `src/emc/task/emccanon.cc`.

emctask

exposes the `emcStatus` class instance. See `src/emc/task/taskmodule.cc`. Not present when using the `gcode` module used for user interfaces - only present in the `milltask` instance of the interpreter.

9.6.10 Adding Predefined Named Parameters

The interpreter comes with a set of predefined named parameters for accessing internal state from the NGC language level. These parameters are read-only and global, and hence cannot be assigned to.

Additional parameters may be added by defining a function in the `namedparams` module. The name of the function defines the name of the new predefined named parameter, which now can be referenced in arbitrary expressions.

To add or redefine a named parameter:

- add a `namedparams` module so it can be found by the interpreter
- define new parameters by functions (see below). These functions receive `self` (the interpreter instance) as parameter and so can access arbitrary state. Arbitrary Python capabilities can be used to return a value.
- import that module from the `TOPLEVEL` script

```
# namedparams.py
# trivial example
def _pi(self):
    return 3.1415926535
```

```
#<circumference> = [2 * #<radius> * #<_pi>]
```

Functions in `namedparams.py` are expected to return a float or int value. If a string is returned, this sets the interpreter error message and aborts execution.

Only functions with a leading underscore are added as parameters, since this is the RS274NGC convention for globals.

It is possible to redefine an existing predefined parameter by adding a Python function of the same name to the `namedparams` module. In this case, a warning is generated during startup.

While the above example isn't terribly useful, note that pretty much all of the interpreter internal state is accessible from Python, so arbitrary predicates may be defined this way. For a slightly more advanced example, see `tests/remap/predefined-named-params`.

9.6.11 Standard Glue routines

Since many remapping tasks are very similar, I've started collecting working prolog and epilog routines in a single Python module. These can currently be found in `ncfiles/remap_lib/python-stdglue/stdglue.py` and provide the following routines:

9.6.11.1 T: prepare_prolog and prepare_epilog

These wrap a NGC procedure for Tx Tool Prepare.

Actions of prepare_prolog The following parameters are made visible to the NGC procedure:

- #<tool> - the parameter of the T word
- #<pocket> - the corresponding pocket

If tool number zero is requested (meaning Tool unload), the corresponding pocket is passed as -1.
It is an error if:

- no tool number is given as T parameter
- the tool cannot be found in the tool table.

Note that unless you set the [EMCIO] RANDOM_TOOLCHANGER=1 parameter, tool and pocket number are identical, and the pocket number from the tool table is ignored. This is currently a restriction.

Actions of prepare_epilog

- The NGC procedure is expected to return a positive value, otherwise an error message containing the return value is given and the interpreter aborts.
- In case the NGC procedure executed the T command (which then refers to the built in T behavior), no further action is taken. This can be used for instance to minimally adjust the built in behavior by preceding or following it with some other statements.
- Otherwise, the #<tool> and #<pocket> parameters are extracted from the subroutine's parameter space. This means that the NGC procedure could change these values, and the epilog takes the changed values in account.
- then, the Canon command SELECT_TOOL(#<tool>) is executed.

9.6.11.2 M6: change_prolog and change_epilog

These wrap a NGC procedure for M6 Tool Change.

Actions of change_prolog

- The following three steps are applicable only if the iocontrol-v2 component is used:
 - If parameter 5600 (fault indicator) is greater than zero, this indicates a Toolchanger fault, which is handled as follows:
 - if parameter 5601 (error code) is negative, this indicates a hard fault and the prolog aborts with an error message.
 - if parameter 5601 (error code) is greater equal zero, this indicates a soft fault. An informational message is displayed and the prolog continues.
- If there was no preceding T command which caused a pocket to be selected, the prolog aborts with an error message.
- If cutter radius compensation is on, the prolog aborts with an error message.

Then, the following parameters are exported to the NGC procedure:

- #<tool_in_spindle> : the tool number of the currently loaded tool
-

- `#<selected_tool>` : the tool number selected
- `#<selected_pocket>` : the selected tool's tooldata index

Actions of `+change_epilog`

- The NGC procedure is expected to return a positive value, otherwise an error message containing the return value is given and the interpreter aborts.
- If parameter 5600 (fault indicator) is greater than zero, this indicates a Toolchanger fault, which is handled as follows (`iocontrol-v2-only`):
 - if parameter 5601 (error code) is negative, this indicates a hard fault and the epilog aborts with an error message.
 - if parameter 5601 (error code) is greater equal zero, this indicates a soft fault. An informational message is displayed and the epilog continues.
- In case the NGC procedure executed the M6 command (which then refers to the built in M6 behavior), no further action is taken. This can be used for instance to minimally adjust the built in behavior by preceding or following it with some other statements.
- Otherwise, the `#<selected_pocket>` parameter is extracted from the subroutine's parameter space, and used to set the interpreter's `current_pocket` variable. Again, the procedure could change this value, and the epilog takes the changed value in account.
- then, the Canon command `CHANGE_T00L(#<selected_pocket>)` is executed.
- The new tool parameters (offsets, diameter etc) are set.

9.6.11.3 G-code Cycles: `cycle_prolog` and `cycle_epilog`

These wrap a NGC procedure so it can act as a cycle, meaning the motion code is retained after finishing execution. If the next line just contains parameter words (e.g. new X,Y values), the code is executed again with the new parameter words merged into the set of the parameters given in the first invocation.

These routines are designed to work in conjunction with an `argspec=<words> parameter`. While this is easy to use, in a realistic scenario you would avoid `argspec` and do a more thorough investigation of the block manually in order to give better error messages.

The suggested `argspec` is as follows:

```
REMAP=G<somecode> argspec=xyzabcuvwqplr prolog=cycle_prolog ngc=<ngc procedure> epilog= ↵
      cycle_epilog modalgroup=1
```

This will permit `cycle_prolog` to determine the compatibility of any axis words given in the block, see below.

Actions of `cycle_prolog`

- Determine whether the words passed in from the current block fulfill the conditions outlined under [Canned Cycle Errors](#).
 - export the axis words as `<x>`, `#<y>` etc; fail if axis words from different groups (XYZ) (UVW) are used together, or any of (ABC) is given.
 - export `L-` as `#<l>`; default to 1 if not given.
 - export `P-` as `#<p>`; fail if `p` less than 0.
 - export `R-` as `#<r>`; fail if `r` not given, or less equal 0 if given.

- fail if feed rate is zero, or inverse time feed or cutter compensation is on.
- Determine whether this is the first invocation of a cycle G-code, if so:
 - Add the words passed in (as per argspec) into a set of sticky parameters, which is retained across several invocations.
- If not (a continuation line with new parameters):
 - merge the words passed in into the existing set of sticky parameters.
- export the set of sticky parameters to the NGC procedure.

Actions of cycle_epilog

- Determine if the current code was in fact a cycle, if so:
 - retain the current motion mode so a continuation line without a motion code will execute the same motion code.

9.6.11.4 S (Set Speed) : setspeed_prolog and setspeed_epilog

TBD

9.6.11.5 F (Set Feed) : setfeed_prolog and setfeed_epilog

TBD

9.6.11.6 M61 Set tool number : settool_prolog and settool_epilog

TBD

9.6.12 Remapped code execution

9.6.12.1 NGC procedure call environment during remaps

Normally, an O-word procedure is called with positional parameters. This scheme is very limiting in particular in the presence of optional parameters. Therefore, the calling convention has been extended to use something remotely similar to the Python keyword arguments model.

see LINKTO gcode/main Subroutines: sub, endsub, return, call.

9.6.12.2 Nested remapped codes

Remapped codes may be nested just like procedure calls - that is, a remapped code whose NGC procedure refers to some other remapped code will execute properly.

The maximum nesting level remaps is currently 10.

9.6.12.3 Sequence number during remaps

Sequence numbers are propagated and restored like with O-word calls. See tests/remap/nested-remaps/w for the regression test, which shows sequence number tracking during nested remaps three levels deep.

9.6.12.4 Debugging flags

The following flags are relevant for remapping and Python - related execution:

EMC_DEBUG_0WORD	0x00002000	traces execution of 0-word subroutines
EMC_DEBUG_REMAP	0x00004000	traces execution of remap-related code
EMC_DEBUG_PYTHON	0x00008000	calls to the Python plug in
EMC_DEBUG_NAMEDPARAM	0x00010000	trace named parameter access
EMC_DEBUG_PYTHON_TASK	0x00040000	trace the task Python plug in
EMC_DEBUG_USER1	0x10000000	user-defined - not interpreted by LinuxCNC
EMC_DEBUG_USER2	0x20000000	user-defined - not interpreted by LinuxCNC

or these flags into the [EMC]DEBUG variable as needed. For a current list of debug flags see *src/emc/nml_intf/debugflags.h*.

9.6.12.5 Debugging Embedded Python code

Debugging of embedded Python code is harder than debugging normal Python scripts, and only a limited supply of debuggers exists. A working open-source based solution is to use the [Eclipse IDE](#), and the [PyDev](#) Eclipse plug in and its [remote debugging feature](#).

To use this approach:

- install Eclipse via the the *Ubuntu Software Center* (choose first selection)
- install the PyDev plug in from the [Pydev Update Site](#)
- setup the LinuxCNC source tree as an Eclipse project
- start the Pydev Debug Server in Eclipse
- make sure the embedded Python code can find the `pydevd.py` module which comes with that plug in - it's buried somewhere deep under the Eclipse install directory. Set the `pydevd` variable in `util.py` to reflect this directory location.
- `import pydevd` in your Python module - see example `util.py` and `remap.py`
- call `pydevd.settrace()` in your module at some point to connect to the Eclipse Python debug server - here you can set breakpoints in your code, inspect variables, step etc as usual.

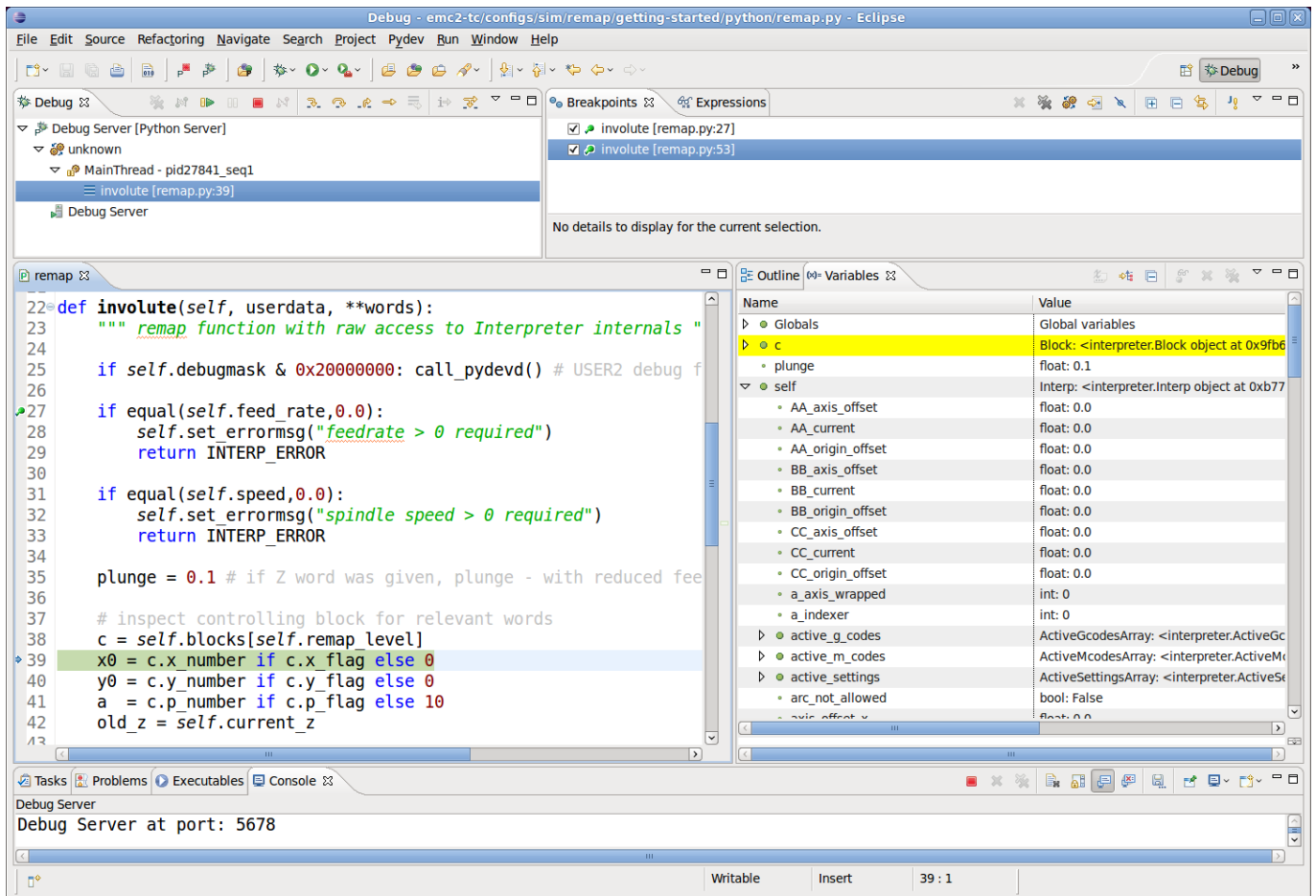


Caution

`pydevd.settrace()` will block execution if Eclipse and the Pydev debug server have not been started.

To cover the last two steps: the `o<pydevd>` procedure helps to get into the debugger from MDI mode. See also the `call_pydevd` function in `util.py` and its usage in `remap.involute` to set a breakpoint.

Here's a screen-shot of Eclipse/PyDevd debugging the `involute` procedure from above:



See the Python code in configs/sim/axis/remap/getting-started/python for details.

9.6.13 Axis Preview and Remapped code execution

For complete preview of a remapped code's tool path some precautions need to be taken. To understand what is going on, let's review the preview and execution process (this covers the Axis case, but others are similar):

First, note that there are **two** independent interpreter instances involved:

- one instance in the milltask program, which executes a program when you hit the *Start* button, and actually makes the machine move
- a second instance in the user interface whose primary purpose is to generate the tool path preview. This one *executes* a program once it is loaded, but it doesn't actually cause machine movements.

Now assume that your remap procedure contains a G38 probe operation, for example as part of a tool change with automatic tool length touch off. If the probe fails, that would clearly be an error, so you'd display a message and abort the program.

Now, what about preview of this procedure? At preview time, of course it's not known whether the probe succeeds or fails - but you would likely want to see what the maximum depth of the probe is, and assume it succeeds and continues execution to preview further movements. Also, there is no point in displaying a *probe failed* message and aborting **during preview**.

The way to address this issue is to test in your procedure whether it executes in preview or execution mode. This can be checked for by testing the `#<_task>` [predefined named parameter](#) - it will be 1

Table 9.3: Table_of_Allocated_G-codes_10-19

#	Gxx	Gxx.1	Gxx.2	Gxx.3	Gxx.4	Gxx.5	Gxx.6	Gxx.7	Gxx.8	Gxx.9
10	G10									
11										
12										
13										
14										
15										
16										
17	G17	G17.1								
18	G18	G18.1								
19	G19	G19.1								

Table 9.4: Table_of_Allocated_G-codes_20-29

#	Gxx	Gxx.1	Gxx.2	Gxx.3	Gxx.4	Gxx.5	Gxx.6	Gxx.7	Gxx.8	Gxx.9
20	G20									
21	G21									
22										
23										
24										
25										
26										
27										
28	G28	G28.1								
29										

Table 9.5: Table_of_Allocated_G-codes_30-39

#	Gxx	Gxx.1	Gxx.2	Gxx.3	Gxx.4	Gxx.5	Gxx.6	Gxx.7	Gxx.8	Gxx.9
30	G30	G30.1								
31										
32										
33	G30	G30.1								
34										
35										
36										
37										
38										
39										

Table 9.6: Table_of_Allocated_G-codes_40-49

#	Gxx	Gxx.1	Gxx.2	Gxx.3	Gxx.4	Gxx.5	Gxx.6	Gxx.7	Gxx.8	Gxx.9
40	G40									
41	G41	G41.1								

Table 9.6: (continued)

[illegible]

Table 9.7: Table_of_Allocated_G-codes_50-59

#	Gxx	Gxx.1	Gxx.2	Gxx.3	Gxx.4	Gxx.5	Gxx.6	Gxx.7	Gxx.8	Gxx.9
50										
51										
52										
53	G53									
54	G54									
55	G55									
56	G56									
57	G57									
58	G58									
59	G59	G59.1	G59.2	G59.3						

Table 9.8: Table of Allocated G-codes 60-69

[illegible]

Table 9.9: Table of Allocated G-codes 70-79

[illegible]

Table 9.9: (continued)

#	Gxx	Gxx.1	Gxx.2	Gxx.3	Gxx.4	Gxx.5	Gxx.6	Gxx.7	Gxx.8	Gxx.9
74										
75										
76	G76									
77										
78										
79										

Table 9.10: Table_of_Allocated_G-codes_80-89

#	Gxx	Gxx.1	Gxx.2	Gxx.3	Gxx.4	Gxx.5	Gxx.6	Gxx.7	Gxx.8	Gxx.9
80	G80									
81	G81									
82	G82									
83	G83									
84	G84									
85	G85									
86	G86									
87	G87									
88	G88									
89	G89									

Table 9.11: Table_of_Allocated_G-codes_90-99

#	Gxx	Gxx.1	Gxx.2	Gxx.3	Gxx.4	Gxx.5	Gxx.6	Gxx.7	Gxx.8	Gxx.9
90	G90	G90.1								
91	G91	G91.1								
92	G92	G92.1	G92.2	G92.3						
93	G93									
94	G94									
95	G95									
96	G96									
97	G97									
98	G98									
99	G99									

9.6.14.3 Currently unallocated M-codes:

These M-codes are currently undefined in the current implementation of LinuxCNC and may be used to define new M-codes. (Developers who define new M-codes in LinuxCNC are encouraged to remove them from this table.)

Table 9.12: Table_of_Unallocated_M-codes_00-99

#	Mx0	Mx1	Mx2	Mx3	Mx4	Mx5	Mx6	Mx7	Mx8	Mx9
00-09										
10-19	M10	M11	M12	M13	M14	M15	M16	M17	M18	
20-29	M20	M21	M22	M23	M24	M25	M26	M27	M28	M29
30-39		M31	M32	M33	M34	M35	M36	M37	M38	M39
40-49	M40	M41	M42	M43	M44	M45	M46	M47		
50-59					M54	M55	M56	M57	M58	M59
60-69										
70-79					M74	M75	M76	M77	M78	M79
80-89	M80	M81	M82	M83	M84	M85	M86	M87	M88	M89
90-99	M90	M91	M92	M93	M94	M95	M96	M97	M98	M99

All M-codes from M100 to M199 are user-defined M-codes already, and should not be remapped.

All M-codes from M200 to M999 are available for remapping.

9.6.14.4 readahead time and execution time

FIXME Write missing information

9.6.14.5 plugin/pickle hack

FIXME Write missing information

9.6.14.6 Module, methods, classes, etc reference

FIXME Write missing information

9.6.15 Introduction: Extending Task Execution

FIXME Write missing information

9.6.15.1 Why would you want to change Task Execution?

FIXME Write missing information

9.6.15.2 A diagram: task, interp, iocontrol, UI (??)

FIXME Write missing information

9.6.16 Models of Task execution

FIXME Write missing information

9.6.16.1 Traditional iocontrol/iocontrolv2 execution

FIXME Write missing information

9.6.16.2 Redefining IO procedures

FIXME Write missing information

9.6.16.3 Execution-time Python procedures

FIXME Write missing information

9.6.17 A short survey of LinuxCNC program execution

To understand remapping of codes it might be helpful to survey the execution of task and interpreter as far as it relates to remapping.

9.6.17.1 Interpreter state

Conceptually, the interpreter's state consist of variables which fall into the following categories:

1. configuration information (typically from INI file)
2. the *World model* - a representation of actual machine state
3. modal state and settings
Refers to state which is *carried over* between executing individual NGC codes - for instance, once the spindle is turned on and the speed is set, it remains at this setting until turned off. The same goes for many codes, like feed, units, motion modes (feed or rapid) and so forth.
4. interpreter execution state
Holds information about the block currently executed, whether we are in a subroutine, interpreter variables etc.
Most of this state is aggregated in a - fairly unsystematic - structure `_setup` (see `interp_internals.hh`).

9.6.17.2 Task and Interpreter interaction, Queuing and Read-Ahead

The task part of LinuxCNC is responsible for coordinating actual machine commands - movement, HAL interactions and so forth. It does not by itself handle the RS274NGC language. To do so, task calls upon the interpreter to parse and execute the next command - either from MDI or the current file.

The interpreter execution generates canonical machine operations, which actually move something. These are, however, not immediately executed but put on a queue. The actual execution of these codes happens in the task part of LinuxCNC: canon commands are pulled off that interpreter queue, and executed resulting in actual machine movements.

This means that typically the interpreter is far ahead of the actual execution of commands - the parsing of the program might well be finished before any noticeable movement starts. This behavior is called *read-ahead*.

9.6.17.3 Predicting the machine position

To compute canonical machine operations in advance during read ahead, the interpreter must be able to predict the machine position after each line of G-code, and that is not always possible.

Let's look at a simple example program which does relative moves (G91), and assume the machine starts at x=0,y=0,z=0. Relative moves imply that the outcome of the next move relies on the position of the previous one:

```
N10 G91
N20 G0 X10 Y-5 Z20
N30 G1 Y20 Z-5
N40 G0 Z30
N50 M2
```

Here the interpreter can clearly predict machine positions for each line:

After N20: x=10 y=-5 z=20; after N30: x=10 y=15 z=15; after N40: x=10 y=15 z=45

and so can parse the whole program and generate canonical operations well in advance.

9.6.17.4 Queue-busters break position prediction

However, complete read ahead is only possible when the interpreter can predict the position impact for **every** line in the program in advance. Let's look at a modified example:

```
N10 G91
N20 G0 X10 Y-5 Z20
N30 G38.3 Z-10
N40 0100 if [#5070 EQ 0]
N50     G1 Y20 Z-5
N60 0100 else
N70     G0 Z30
N80 0100 endif
N90 G1 Z10
N95 M2
```

To pre-compute the move in N90, the interpreter would need to know where the machine is after line N80 - and that depends on whether the probe command succeeded or not, which is not known until it's actually executed.

So, some operations are incompatible with further read-ahead. These are called *queue busters*, and they are:

- reading a HAL pin's value with M66: value of HAL pin not predictable
- loading a new tool with M6: tool geometry not predictable
- executing a probe with G38.n: final position and success/failure not predictable

9.6.17.5 How queue-busters are dealt with

Whenever the interpreter encounters a queue-buster, it needs to stop read ahead and wait until the relevant result is available. The way this works is:

- when such a code is encountered, the interpreter returns a special return code to task (*INTERP_EXECUTE*).
- this return code signals to task to stop read ahead for now, execute all queued canonical commands built up so far (including the last one, which is the queue buster), and then *synchronize the interpreter state with the world model*. Technically, this means updating internal variables to reflect HAL pin values, reload tool geometries after an M6, and convey results of a probe.

- The interpreter's *synch()* method is called by task and does just that - read all the world model *actual* values which are relevant for further execution.
- at this point, task goes ahead and calls the interpreter for more read ahead - until either the program ends or another queue-buster is encountered.

9.6.17.6 Word order and execution order

One or several *words* may be present on an NGC *block* if they are compatible (some are mutually exclusive and must be on different lines). The execution model however prescribes a strict ordering of execution of codes, regardless of their appearance on the source line ([G-code Order of Execution](#)).

9.6.17.7 Parsing

Once a line is read (in either MDI mode, or from the current NGC file), it is parsed and flags and parameters are set in a *struct block* (struct_setup, member block1). This struct holds all information about the current source line, but independent of different ordering of codes on the current line: as long as several codes are compatible, any source ordering will result in the same variables set in the struct block. Right after parsing, all codes on a block are checked for compatibility.

9.6.17.8 Execution

After successful parsing the block is executed by `execute_block()`, and here the different items are handled according to execution order.

If a "queue buster" is found, a corresponding flag is set in the interpreter state (toolchange_flag, input_flag, probe_flag) and the interpreter returns an `INTERP_EXECUTE_FINISH` return value, signaling *stop readahead for now, and resynch* to the caller (*task*). If no queue busters are found after all items are executed, `INTERP_OK` is returned, signalling that read-ahead may continue.

When read ahead continues after the synch, task starts executing interpreter `read()` operations again. During the next read operation, the above mentioned flags are checked and corresponding variables are set (because the a `synch()` was just executed, the values are now current). This means that the next command already executes in the properly set variable context.

9.6.17.9 Procedure execution

O-word procedures complicate handling of queue busters a bit. A queue buster might be found somewhere in a nested procedure, resulting in a semi-finished procedure call when `INTERP_EXECUTE_FINISH` is returned. Task makes sure to synchronize the world model, and continue parsing and execution as long as there is still a procedure executing (`call_level > 0`).

9.6.17.10 How tool change currently works

The actions happening in LinuxCNC are a bit involved, but it's necessary to get the overall idea what currently happens before you set out to adapt those workings to your own needs.

Note that remapping an existing code completely disables all internal processing for that code. That means that beyond your desired behavior - probably described through an NGC Oword or Python procedure, you need to replicate those internal actions of the interpreter which together result in a complete replacement of the existing code. The prolog and epilog code is the place to do this.

How tool information is communicated Several processes are *interested* in tool information: task and its interpreter, as well as the user interface. Also, the *halui* process.

Tool information is held in the *emcStatus* structure, which is shared by all parties. One of its fields is the *toolTable* array, which holds the description as loaded from the tool table file (tool number, diameter, frontangle, backangle and orientation for lathe, tool offset information).

The authoritative source and only process actually *setting* tool information in this structure is the *iocontrol* process. All others processes just consult this structure. The interpreter holds actually a local copy of the tool table.

For the curious, the current *emcStatus* structure can be accessed by [Python statements](#). The interpreter's perception of the tool currently loaded for instance is accessed by:

```
;py,from interpreter import *
;py,print(this.tool_table[0])
```

To see fields in the global *emcStatus* structure, try this:

```
;py,from emctask import *
;py,print(emcstat.io.tool.pocketPrepped)
;py,print(emcstat.io.tool.toolInSpindle)
;py,print(emcstat.io.tool.toolTable[0])
```

You need to have LinuxCNC started from a terminal window to see the results.

9.6.17.11 How Tx (Prepare Tool) works

Interpreter action on a Tx command

All the interpreter does is evaluate the *toolnumber* parameter, looks up its corresponding *tooldata* index, remembers it in the *selected_pocket* variable for later, and queues a canon command (*SELECT_TOOL*). See *Interp::convert_tool_select* in *src/emc/rs274/interp_execute.cc*.

Task action on SELECT_TOOL When task gets around to handle a *SELECT_TOOL*, it sends a *EMC_TOOL_P* message to the *iocontrol* process, which handles most tool-related actions in LinuxCNC.

In the current implementation, task actually waits for *iocontrol* to complete the changer positioning operation, which is not necessary IMO since it defeats the idea that changer preparation and code execution can proceed in parallel.

Iocontrol action on EMC_TOOL_PREPARE When *iocontrol* sees the select pocket command, it does the related HAL pin wiggling - it sets the "tool-prep-number" pin to indicate which tool is next, raises the "tool-prepare" pin, and waits for the "tool-prepared" pin to go high.

When the changer responds by asserting "tool-prepared", it considers the prepare phase to be completed and signals task to continue. (again, this *wait* isn't strictly necessary IMO)

Building the prolog and epilog for Tx See the Python functions *prepare_prolog* and *prepare_epilog* in *configs/sim/axis/remap/toolchange/python/toolchange.py*.

9.6.17.12 How M6 (Change tool) works

You need to understand this fully before you can adapt it. It is very relevant to writing a prolog and epilog handler for a remapped M6. Remapping an existing codes means you disable the internal steps taken normally, and replicate them as far as needed for your own purposes.

Even if you are not familiar with C, I suggest you look at the *Interp::convert_tool_change* code in *src/emc/rs274/interp_convert.cc*.

Interpreter action on a M6 command

When the interpreter sees an M6, it:

1. checks whether a T command has already been executed (test *settings->selected_pocket* to be ≥ 0) and fail with *Need tool prepared -Txx- for toolchange* message if not.
2. check for cutter compensation being active, and fail with *Cannot change tools with cutter radius compensation on* if so.
3. stop the spindle except if the "TOOL_CHANGE_WITH_SPINDLE_ON" ini option is set.
4. generate a rapid Z up move if if the "TOOL_CHANGE_QUILL_UP" ini option is set.
5. if TOOL_CHANGE_AT_G30 was set:
 - a. move the A, B and C indexers if applicable
 - b. generate rapid move to the G30 position
6. execute a CHANGE_TOOL canon command, with the selected pocket as parameter. CHANGE_TOOL will:
 - a. generate a rapid move to TOOL_CHANGE_POSITION if so set in ini
 - b. enqueue an EMC_TOOL_LOAD NML message to task.
7. set the numberer parameters 5400-5413 according to the new tool
8. signal to task to stop calling the interpreter for readahead by returning INTERP_EXECUTE_FINISH since M6 is a queue buster.

What task does when it sees a CHANGE_TOOL command Again, not much more than passing the buck to iocontrol by sending it an EMC_TOOL_LOAD message, and waiting until iocontrol has done its thing.

Iocontrol action on EMC_TOOL_LOAD

1. it asserts the "tool-change" pin
2. it waits for the "tool-changed" pin to become active
3. when that has happened:
 - a. deassert "tool-change"
 - b. set "tool-prep-number" and "tool-prep-pocket" pins to zero
 - c. execute the *load_tool()* function with the pocket as parameter.

The last step actually sets the tooltable entries in the *emcStatus* structure. The actual action taken depends on whether the RANDOM_TOOLCHANGER ini option was set, but at the end of the process *toolTable[0]* reflects the tool currently in the spindle.

When that has happened:

1. iocontrol signals task to go ahead
2. task tells the interpreter to execute a *synch()* operation, to see what has changed
3. the interpreter *synch()* pulls all information from the world model needed, among it the changed tool table.

From there on, the interpreter has complete knowledge of the world model and continues with read ahead.

Building the prolog and epilog for M6 See the Python functions *change_prolog* and *change_epilog* in *configs/sim/axis/remap/toolchange/python/toolchange.py*.

9.6.17.13 How M61 (Change tool number) works

M61 requires a non-negative `Q` parameter (tool number). If zero, this means *unload tool*, else *set current tool number to Q*.

Building the replacement for M61 An example Python redefinition for M61 can be found in the `set_tool_number` function in `configs/sim/axis/remap/toolchange/python/toolchange.py`.

9.6.18 Status

1. the RELOAD_ON_CHANGE feature is fairly broken. Restart after changing a Python file.
2. M61 (remapped or not) is broken in iocontrol and requires iocontrol-v2 to actually work.

9.6.19 Changes

- the method to return error messages and fail used to be `self.set_errormsg(text)` followed by `return INTERP_ERROR`. This has been replaced by merely returning a string from a Python handler or oword subroutine. This sets the error message and aborts the program. Previously there was no clean way to abort a Python oword subroutine.

9.6.20 Debugging

In the `[EMC]` section of the ini file the `DEBUG` parameter can be changed to get various levels of debug messages when LinuxCNC is started from a terminal.

```
Debug level, 0 means no messages. See src/emc/nml_intf/debugflags.h for others
DEBUG = 0x00000002 # configuration
DEBUG = 0x7FFFDEFF # no interp,oword
DEBUG = 0x00008000 # py only
DEBUG = 0x0000E000 # py + remap + 0word
DEBUG = 0x0000C002 # py + remap + config
DEBUG = 0x0000C100 # py + remap + Interpreter
DEBUG = 0x0000C140 # py + remap + Interpreter + NML msgs
DEBUG = 0x0000C040 # py + remap + NML
DEBUG = 0x0003E100 # py + remap + Interpreter + oword + signals + namedparams
DEBUG = 0x10000000 # EMC_DEBUG_USER1 - trace statements
DEBUG = 0x20000000 # EMC_DEBUG_USER2 - trap into Python debugger
DEBUG = 0x10008000 # USER1, PYTHON
DEBUG = 0x30008000 # USER1,USER2, PYTHON # USER2 will cause involute to try to connect to ↔
pydev
DEBUG = 0x7FFFFFFF # All debug messages
```

9.7 Moveoff Component

The moveoff HAL component is a HAL-only method for implementing offsets. See the manpage (`$ man moveoff`) for the IMPORTANT limitations and warnings.

The moveoff component is used to offset joint positions using custom HAL connections. Implementing an offset-while-program-is-paused functionality is supported with appropriate connections for the input pins. Nine joints are supported.

The axis offset pin values (offset-in-M) are continuously applied (respecting limits on value, velocity, and acceleration) to the output pins (offset-current-M, pos-plusoffset-M, fb-minusoffset-M) when both

enabling input pins (apply-offsets and move-enable) are TRUE. The two enabling inputs are anded internally. A *warning pin* is set and a message issued if the apply-offsets pin is deasserted while offsets are applied. The warning pin remains TRUE until the offsets are removed or the apply-offsets pin is set.

Typically, the move-enable pin is connected to external controls and the apply-offsets pin is connected to halui.program.is-paused (for offsets only while paused) or set to TRUE (for continuously applied offsets).

Applied offsets are *automatically returned* to zero (respecting limits) when either of the enabling inputs is deactivated. The zero value tolerance is specified by the epsilon input pin value.

Waypoints are recorded when the moveoff component is enabled. Waypoints are managed with the waypoint-sample-secs and waypoint-threshold pins. When the backtrack-enable pin is TRUE, the auto-return path follows the recorded waypoints. When the memory available for waypoints is exhausted, offsets are frozen and the waypoint-limit pin is asserted. This restriction applies regardless of the state of the backtrack-enable pin. An enabling pin must be deasserted to allow a return to the original (non-offset position).

Backtracking through waypoints results in *slower* movement rates as the moves are point-to-point respecting velocity and acceleration settings. The velocity and acceleration limit pins can be managed dynamically to control offsets at all times.

When backtrack-enable is FALSE, the auto-return move is **NOT** coordinated, each axis returns to zero at its own rate. If a controlled path is wanted in this condition, each axis should be manually returned to zero before deasserting an enabling pin.

The waypoint-sample-secs, waypoint-threshold, and epsilon pins are evaluated only when the component is idle.

The offsets-applied output pin is provided to indicate the current state to a GUI so that program resumption can be managed. If the offset(s) are non-zero when the apply-offsets pin is deasserted (for example when resuming a program when offsetting during a pause), offsets are returned to zero (respecting limits) and an *Error* message is issued.



Caution

If offsets are enabled and applied and the machine is turned off for any reason, any *external* HAL logic that manages the enabling pins and the offset-in-M inputs is responsible for their state when the machine is subsequently turned on again.

This HAL-only means of offsetting is typically not known to LinuxCNC nor available in GUI preview displays. **No protection is provided** for offset moves that exceed soft limits managed by LinuxCNC. Since soft limits are not honored, an offset move may encounter hard limits (or **CRASH** if there are no limit switches). Use of the offset-min-M and offset-max-M inputs to limit travel is recommended. Triggering a hard limit will turn off the machine — see **Caution** above.

The offset-in-M values may be set with INI file settings, controlled by a GUI, or managed by other HAL components and connections. Fixed values may be appropriate in simple cases where the direction and amount of offset is well-defined but a control method is required to deactivate an enabling pin in order to return offsets to zero. GUIs may provide means for users to set, increment, decrement, and accumulate offset values for each axis and may set offset-in-M values to zero before deasserting an enabling pin.

The default values for accel, vel, min, max, epsilon, waypoint-sample-secs, and waypoint-threshold may not be suitable for any particular application. This HAL component is unaware of limits enforced elsewhere by LinuxCNC. Users should test usage in a simulator application and understand all hazards before use on hardware.

Sim configurations that demonstrate the component and a GUI (moveoff_gui) are located in:

- configs/sim/axis/moveoff (axis-ui)
 - configs/sim/touchy/ngcgui (touchy-ui)
-

9.7.1 Modifying an existing configuration

A system-provided HAL file (LIB:hookup_moveoff.tcl) can be used to adapt an existing configuration to use the moveoff component. Additional INI file settings support the use of a simple GUI (moveoff_gui) for controlling offsets.

When the system HAL file (LIB:hookup_moveoff.tcl) is properly specified in a configuration INI file, it will:

1. Disconnect the original joint.N.motor-pos-cmd and joint.N.motor-pos-fb pin connections
2. Load (loadrt) the moveoff component (using the name mv) with a personality set to accommodate all axes identified in the INI file
3. Add (addf) the moveoff component functions in the required sequence
4. Reconnect the joint.N.motor-pos-cmd and joint.N.motor-pos-fb pins to use the moveoff component
5. Set the moveoff component operating parameters and limits for each axis in accordance with additional INI file settings

Note: The moveoff_gui application supports configurations that use known kinematics modules with KINEMATICS_TYPE=KINEMATICS_IDENTITY. Supported modules include: trivkins. With identity kins, moveoff_gui assigns each axis name specified with the command line parameter *-axes axisnames* to the corresponding joint.

Modify an existing configuration as follows:

Make sure there is an INI file entry for [HAL]HALUI and create a new [HAL]HALFILE entry for LIB:hookup_moveoff.tcl. The entry for LIB:hookup_moveoff.tcl should follow all HALFILE= entries for HAL files that connect the pins for joint.N.motor-pos-cmd, joint.N.motor-pos-fb, and any components connected to these pins (pid and encoder components in a servo system for instance).

```
[HAL]
HALUI    = halui
HALFILE  = existing_configuration_halfile_1
...
HALFILE  = existing_configuration_halfile_n
HALFILE  = LIB:hookup_moveoff.tcl
```

Add INI file entries for the per-axis settings for each axis in use (if an entry is not defined, the corresponding entry from the [AXIS_n] section will be used, if no entry is found, then the moveoff component default will be used).

Note

Using component defaults or [AXIS_n] section values for per-axis offset settings is NOT recommended.

```
[MOVEOFF_n]
MAX_LIMIT =
MIN_LIMIT =
MAX_VELOCITY =
MAX_ACCELERATION =
```

Add INI file entries for moveoff component settings (omit to use moveoff defaults):

```
[MOVEOFF]
EPSILON =
WAYPOINT_SAMPLE_SECS =
WAYPOINT_THRESHOLD =
```

The `moveoff_gui` is used to make additional required connections and provide a popup GUI to:

1. Provide a control togglebutton to Enable/Disable offsets
2. Provide a control togglebutton to Enable/Disable backtracking
3. Provide control pushbuttons to Increment/Decrement/Zero each axis offset
4. Display each axis offset current value
5. Display current offset status (disabled, active, removing, etc)

The provided control buttons are optional depending upon the state of the `moveoff` component `move-enable` pin. Both a display and controls for enabling offsetting are provided if the pin `mv.move-enable` is NOT connected when the `moveoff_gui` is started. For this case, the `moveoff_gui` manages the `moveoff` component `move-enable` pin (named `mv.move-enable`) as well as the offsets (`mv.move-offset-in-M`) and the backtracking enable (`mv.backtrack-enable`)

If the `mv.move-enable` pin IS connected when the `moveoff_gui` is started, the `moveoff_gui` will provide a display but NO controls. This mode supports configurations that use a jog wheel or other methods of controlling the offset inputs and the enable pins (`mv.offset-in-M`, `mv.move-enable`, `mv.backtrack-enable`).

The `moveoff_gui` makes the required connections for the `moveoff` component pins: `mv.power_on` and `mv.apply-offsets`. The `mv.power_on` pin is connected to the `motion.motion-enabled` pin (a new signal is automatically created if necessary). The `mv.apply-offsets` is connected to `halui.program.is-paused` or set to 1 depending upon the command line option `-mode [onpause | always]`. A new signal is automatically created if necessary.

To use the `moveoff_gui`, add an entry in the INI file [APPLICATIONS] section as follows:

```
[APPLICATIONS]
# Note: a delay (specified in seconds) may be required if connections
# are made using postgui HAL files ([HAL]POSTGUI_HALFILE=)
DELAY = 0
APP = moveoff_gui option1 option2 ...
```

When the HAL file `LIB:hookup_moveoff.tcl` is used to load and connect the `moveoff` component, the `mv.move-enable` pin will not be connected and local controls provided by the `moveoff_gui` will be used. This is the simplest method to test or demonstrate the `moveoff` component when modifying an existing INI configuration.

To enable external controls while using the `moveoff_gui` display for offset values and status, HAL files that follow `LIB:hookup_moveoff.tcl` must make additional connections. For example, the supplied demonstration configs (`configs/sim/axis/moveoff/*.ini`) use a simple system HAL file (named `LIB:moveoff_external.hal`) to connect the `mv.move-enable`, `mv.offset-in-M`, and `mv.backtrack-enable` pins to signals:

```
[HAL]
HALUI = halui
...
HALFILE = LIB:hookup_moveoff.tcl
HALFILE = LIB:moveoff_external.hal
```

The connections made by `LIB:moveoff_external.hal` (for a three axis configuration) are:

```
net external_enable mv.move-enable

net external_offset_0 mv.offset-in-0
net external_offset_1 mv.offset-in-1
net external_offset_2 mv.offset-in-2

net external_backtrack_en mv.backtrack-enable
```

These signals (`external_enable`, `external_offset_M`, `external_backtrack_en`) may be managed by subsequent HALFILES (including POSTGUI_HALFILES) to provide customized control of the component while using the `moveoff_gui` display for current offset values and offset status.

The `moveoff_gui` is configured with command line options. For details on the operation of `moveoff_gui`, see the man page:

```
$ man moveoff_gui
```

For a brief listing of command line options for `moveoff_gui`, use the command line help option:

```
$ moveoff_gui --help
```

Usage:

```
moveoff_gui [Options]
```

Options:

```
  [--help | -? | -- -h ] (This text)
```

```
  [-mode [onpause | always]] (default: onpause)
                             (onpause: show gui when program paused)
                             (always:  show gui always)
```

```
  [-axes axisnames]          (default: xyz (no spaces))
                             (letters from set of: x y z a b c u v w)
                             (example: -axes z)
                             (example: -axes xz)
                             (example: -axes xyz)
```

```
  [-inc incrementvalue]      (default: 0.001 0.01 0.10 1.0 )
                             (specify one per -inc (up to 4) )
                             (example: -inc 0.001 -inc 0.01 -inc 0.1 )
```

```
  [-size integer]            (default: 14)
                             (Overall gui popup size is based on font size)
```

```
  [-loc center|+x+y]         (default: center)
                             (example: -loc +10+200)
```

```
  [-autoresume]              (default: notused)
                             (resume program when move-enable deasserted)
```

```
  [-delay delay_secs]        (default: 5 (resume delay))
```

Options for special cases:

```
  [-noentry]                 (default: notused)
                             (don't create entry widgets)
```

```
  [-no_resume_inhibit]       (default: notused)
                             (do not use a resume-inhibit-pin)
```

```
  [-no_pause_requirement]    (default: notused)
                             (no check for halui.program.is-paused)
```

```
  [-no_cancel_autoresume]    (default: notused)
                             (useful for retract offsets with simple)
                             (external control)
```

```
  [-no_display]              (default: notused)
                             (Use when both external controls and displays)
                             (are in use (see Note))
```

Note: If the `moveoff` `move-enable` pin (`mv.move-enable`) is connected when `moveoff_gui` is started, external controls are required and only displays are provided.

9.8 Stepper Configuration

9.8.1 Introduction

The preferred way to set up a standard stepper machine is with the Step Configuration Wizard. See the [Stepper Configuration Wizard](#) Chapter.

This chapter describes some of the more common settings for manually setting up a stepper based system. These systems are using stepper motors with drives that accept step & direction signals.

It is one of the simpler setups, because the motors run open-loop (no feedback comes back from the motors), yet the system needs to be configured properly so the motors don't stall or lose steps.

Most of this chapter is based on a sample config released along with LinuxCNC. The config is called `stepper_inch`, and can be found by running the [Configuration Picker](#).

9.8.2 Maximum step rate

With software step generation, the maximum step rate is one step per two `BASE_PERIOD`s for step-and-direction output. The maximum requested step rate is the product of an axis' `MAX_VELOCITY` and its `INPUT_SCALE`. If the requested step rate is not attainable, following errors will occur, particularly during fast jogs and G0 moves.

If your stepper driver can accept quadrature input, use this mode. With a quadrature signal, one step is possible for each `BASE_PERIOD`, doubling the maximum step rate.

The other remedies are to decrease one or more of: the `BASE_PERIOD` (setting this too low will cause the machine to become unresponsive or even lock up), the `INPUT_SCALE` (if you can select different step sizes on your stepper driver, change pulley ratios, or leadscrew pitch), or the `MAX_VELOCITY` and `STEPGEN_MAXVEL`.

If no valid combination of `BASE_PERIOD`, `INPUT_SCALE`, and `MAX_VELOCITY` is acceptable, then consider using hardware step generation (such as with the LinuxCNC-supported Universal Stepper Controller, Mesa cards, and others).

9.8.3 Pinout

One of the major flaws in LinuxCNC was that you couldn't specify the pinout without recompiling the source code. LinuxCNC is far more flexible, and now (thanks to the Hardware Abstraction Layer) you can easily specify which signal goes where. See the [HAL Basics](#) for more information on HAL.

As it is described in the HAL Introduction and tutorial, we have signals, pins and parameters inside the HAL.

Note

We are presenting one axis to keep it short, all others are similar.

The ones relevant for our pinout are:

```
signals: Xstep, Xdir & Xen  
pins: parport.0.pin-XX-out & parport.0.pin-XX-in
```

Depending on what you have chosen in your INI file you are using either `standard_pinout.hal` or `xy-lotex_pinout.hal`. These are two files that instruct the HAL how to link the various signals & pins. Further on we'll investigate the `standard_pinout.hal`.

9.8.3.1 Standard Pinout HAL

This file contains several HAL commands, and usually looks like this:

```
# standard pinout config file for 3-axis steppers
# using a parport for I/O
#
# first load the parport driver
loadrt hal_parport cfg="0x0378"
#
# next connect the parport functions to threads
# read inputs first
addf parport.0.read base-thread 1
# write outputs last
addf parport.0.write base-thread -1
#
# finally connect physical pins to the signals
net Xstep => parport.0.pin-03-out
net Xdir  => parport.0.pin-02-out
net Ystep => parport.0.pin-05-out
net Ydir  => parport.0.pin-04-out
net Zstep => parport.0.pin-07-out
net Zdir  => parport.0.pin-06-out

# create a signal for the estop loopback
net estop-loop iocontrol.0.user-enable-out iocontrol.0.emc-enable-in

# create signals for tool loading loopback
net tool-prep-loop iocontrol.0.tool-prepare iocontrol.0.tool-prepared
net tool-change-loop iocontrol.0.tool-change iocontrol.0.tool-changed

# connect "spindle on" motion controller pin to a physical pin
net spindle-on spindle.0.on => parport.0.pin-09-out

###
### You might use something like this to enable chopper drives when machine ON
### the Xen signal is defined in core_stepper.hal
###
# net Xen => parport.0.pin-01-out

###
### If you want active low for this pin, invert it like this:
###
# setp parport.0.pin-01-out-invert 1

###
### A sample home switch on the X axis (axis 0).  make a signal,
### link the incoming parport pin to the signal, then link the signal
### to LinuxCNC's axis 0 home switch input pin.
###
# net Xhome parport.0.pin-10-in => joint.0.home-sw-in

###
### Shared home switches all on one parallel port pin?
### that's ok, hook the same signal to all the axes, but be sure to
### set HOME_IS_SHARED and HOME_SEQUENCE in the INI file.
###
# net homeswitches <= parport.0.pin-10-in
```

```
# net homeswitches => joint.0.home-sw-in
# net homeswitches => joint.1.home-sw-in
# net homeswitches => joint.2.home-sw-in

###
### Sample separate limit switches on the X axis (axis 0)
###

# net X-neg-limit parport.0.pin-11-in => joint.0.neg-lim-sw-in
# net X-pos-limit parport.0.pin-12-in => joint.0.pos-lim-sw-in

###
### Just like the shared home switches example, you can wire together
### limit switches. Beware if you hit one, LinuxCNC will stop but can't tell
### you which switch/axis has faulted. Use caution when recovering from this
### extreme position to avoid a hard stop.
###

# net Xlimits parport.0.pin-13-in => joint.0.neg-lim-sw-in joint.0.pos-lim-sw-in
```

The lines starting with `#` are comments, and their only purpose is to guide the reader through the file.

9.8.3.2 Overview

There are a couple of operations that get executed when the `standard_pinout.hal` gets executed/interpreted:

- The Parport driver gets loaded (see the [Parport Chapter](#) for details).
- The read & write functions of the parport driver get assigned to the base thread ¹.
- The step & direction signals for axes X, Y, Z get linked to pins on the parport.
- Further I/O signals get connected (estop loopback, toolchanger loopback).
- A spindle-on signal gets defined and linked to a parport pin.

9.8.3.3 Changing the `standard_pinout.hal`

If you want to change the `standard_pinout.hal` file, all you need is a text editor. Open the file and locate the parts you want to change.

If you want for example to change the pin for the X-axis Step & Directions signals, all you need to do is to change the number in the `parport.0.pin-XX-out` name:

```
net Xstep parport.0.pin-03-out
net Xdir  parport.0.pin-02-out
```

can be changed to:

```
net Xstep parport.0.pin-02-out
net Xdir  parport.0.pin-03-out
```

or basically any other *out* pin you like.

Hint: make sure you don't have more than one signal connected to the same pin.

¹the fastest thread in the LinuxCNC setup, usually the code gets executed every few tens of microseconds

9.8.3.4 Changing polarity of a signal

If external hardware expects an "active low" signal, set the corresponding *-invert* parameter. For instance, to invert the spindle control signal:

```
setp parport.0.pin-09-invert TRUE
```

9.8.3.5 Adding PWM Spindle Speed Control

If your spindle can be controlled by a PWM signal, use the *pwmgen* component to create the signal:

```
loadrt pwmgen output_type=0
addf pwmgen.update servo-thread
addf pwmgen.make-pulses base-thread
net spindle-speed-cmd spindle.0.speed-out => pwmgen.0.value
net spindle-on spindle.0.on => pwmgen.0.enable
net spindle-pwm pwmgen.0.pwm => parport.0.pin-09-out
setp pwmgen.0.scale 1800 # Change to your spindle's top speed in RPM
```

This assumes that the spindle controller's response to PWM is simple: 0% PWM gives 0 RPM, 10% PWM gives 180 RPM, etc. If there is a minimum PWM required to get the spindle to turn, follow the example in the *nist-lathe* sample configuration to use a *scale* component.

9.8.3.6 Adding an enable signal

Some amplifiers (drives) require an enable signal before they accept and command movement of the motors. For this reason there are already defined signals called *Xen*, *Yen*, *Zen*.

To connect them use the following example:

```
net Xen parport.0.pin-08-out
```

You can either have one single pin that enables all drives; or several, depending on the setup you have. Note, however, that usually when one axis faults, all the other drives will be disabled as well, so having only one enable signal / pin for all drives is a common practice.

9.8.3.7 External ESTOP button

The standard *pinout.hal* file assumes no external ESTOP button. For more information on an external E-Stop see the *estop_latch* man page.

Chapter 10

Control Panels

10.1 PyVCP

10.1.1 Introduction

PyVCP, **P**ython **V**irtual **C**ontrol **P**anel, is designed to give the integrator the ability to customize the AXIS interface with buttons and indicators to do special tasks.

Hardware machine control panels can use up a lot of I/O pins and can be expensive. That is where Virtual Control Panels have the advantage as well as it cost nothing to build a PyVCP.

Virtual Control Panels can be used for testing or monitoring things to temporarily replace real I/O devices while debugging ladder logic, or to simulate a physical panel before you build it and wire it to an I/O board.

The following graphic displays many of the PyVCP widgets.



Figure 10.1: PyVCP Widgets Showcase

10.1.2 Panel Construction

The layout of a PyVCP panel is specified with an XML file that contains widget tags between `<pyvcp>` and `</pyvcp>`. For example:

```
<pyvcp>
  <label text="This is a LED indicator"/>
  <led/>
</pyvcp>
```

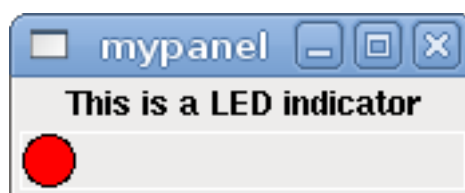


Figure 10.2: Simple PyVCP LED Panel Example

If you place this text in a file called `tiny.xml`, and run

```
halcmd loadusr pyvcp -c mypanel tiny.xml
```

PyVCP will create the panel for you, which includes two widgets, a Label with the text *This is a LED indicator*, and a LED, used for displaying the state of a HAL BIT signal. It will also create a HAL component named *mypanel* (all widgets in this panel are connected to pins that start with *mypanel.*). Since no `<halpin>` tag was present inside the `<led>` tag, PyVCP will automatically name the HAL pin for the LED widget `mypanel.led.0`

For a list of widgets and their tags and options, see the [widget reference](#) below.

Once you have created your panel, connecting HAL signals to and from the PyVCP pins is done with the `halcmd`:

```
net <signal-name> <pin-name> <opt-direction> <opt-pin-name>signal-name
```

If you are new to HAL, the HAL basics chapter in the Integrator Manual is a good place to start.

10.1.3 Security

Parts of PyVCP files are evaluated as Python code, and can take any action available to Python programs. Only use PyVCP .xml files from a source that you trust.

10.1.4 AXIS

Since AXIS uses the same GUI toolkit (Tkinter) as PyVCP, it is possible to include a PyVCP panel at either the right side or the bottom of the AXIS user interface. A typical example is explained below.

Place your PyVCP XML file describing the panel in the same directory where your .ini file is. Say we want to display the current spindle speed using a Bar widget. Place the following in a file called `spindle.xml`:

```
<pyvcp>
  <label>
    <text>"Spindle speed:"</text>
  </label>
  <bar>
    <halpin>"spindle-speed"</halpin>
    <max_>5000</max_>
  </bar>
</pyvcp>
```

Here we've made a panel with a Label and a Bar widget, specified that the HAL pin connected to the Bar should be named *spindle-speed*, and set the maximum value of the bar to 5000 (see [widget reference](#) below for all options). To make AXIS aware of this file, and call it at start up, we need to specify the following in the [DISPLAY] section of the .ini file:

```
PYVCP = spindle.xml
```

If the panel should appear at the bottom of the AXIS user interface then we need to specify the following in the [DISPLAY] section of the .ini file:

```
PYVCP_POSITION = BOTTOM
```

Anything other than `BOTTOM` or omitting this variable will place the PYVCP panel at the right.

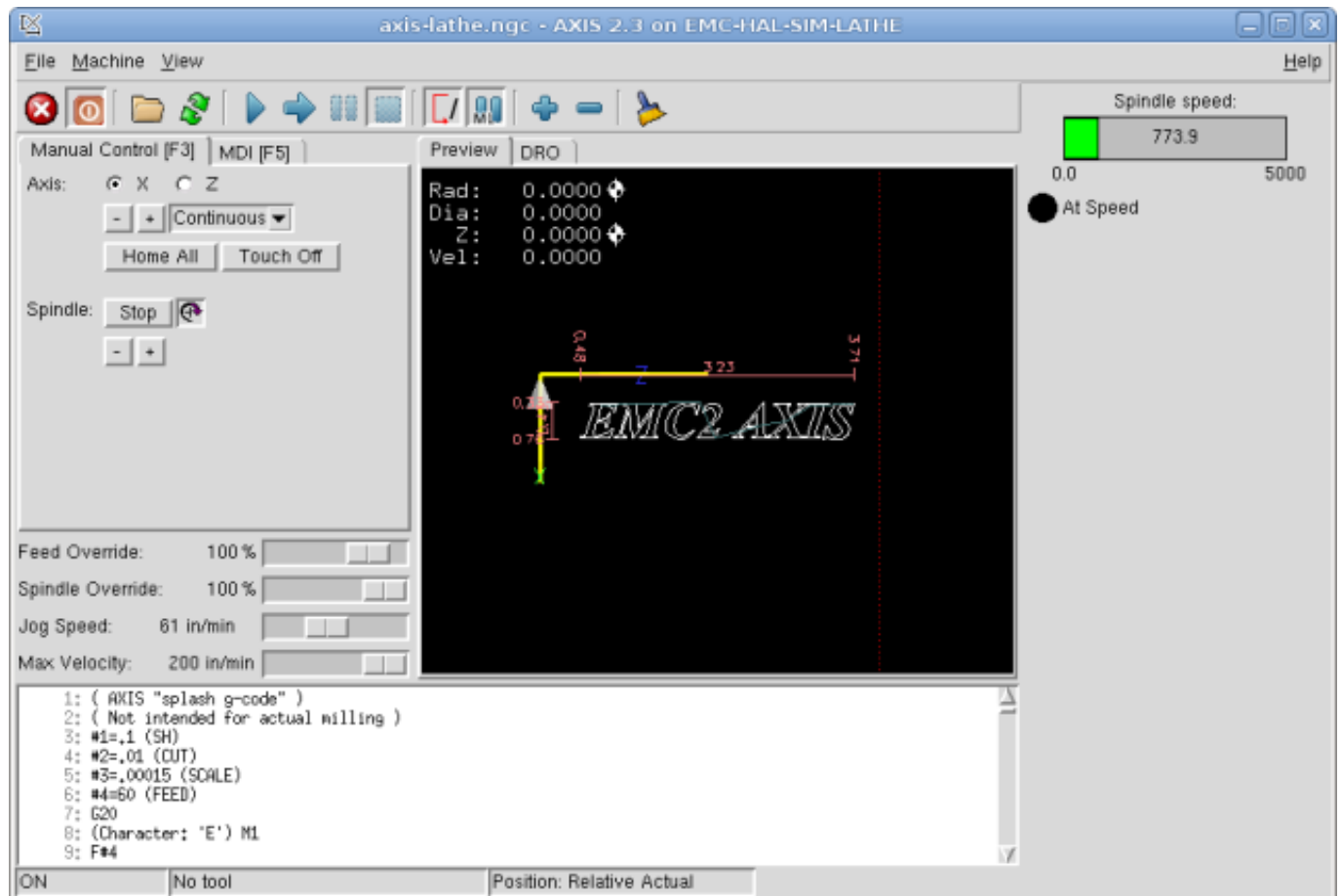
To make our widget actually display the spindle-speed it needs to be hooked up to the appropriate HAL signal. A .hal file that will be run once AXIS and PyVCP have started can be specified in the [HAL] section of the .ini file:

```
POSTGUI_HALFILE = spindle_to_pyvcp.hal
```

This change will run the HAL commands specified in *spindle_to_pyvcp.hal*. In our example the contents could look like this:

```
net spindle-rpm-filtered => pyvcp.spindle-speed
```

assuming that a signal called *spindle-rpm-filtered* already exists. Note that when running together with AXIS, all PyVCP widget HAL pins have names that start with *pyvcp..*



This is what the newly created PyVCP panel should look like in AXIS. The *sim/lathe* configuration is already configured this way.

10.1.5 Stand Alone

This section describes how PyVCP panels can be displayed on their own with or without LinuxCNC's machine controller.

To load a stand alone PyVCP panel with LinuxCNC use these commands:

```
loadusr -Wn mypanel pyvcp -g WxH+X+Y -c mypanel <path/>panel_file.xml
```

You would use this if you wanted a floating panel or a panel with a GUI other than AXIS.

- *-Wn panelname* - makes HAL wait for the component *panelname* to finish loading (*become ready* in HAL speak) before processing more HAL commands. This is important because PyVCP panels export HAL pins, and other HAL components will need them present to connect to them. Note the capital W and lowercase n. If you use the *-Wn* option you must use the *-c* option to name the panel.
- *pyvcp <-g> <-c> panel.xml* - builds the panel with the optional geometry and/or panelname from the xml panel file. The *panel.xml* can be any name that ends in *.xml*. The *.xml* file is the file that describes how to build the panel. You must add the path name if the panel is not in the directory that the HAL script is in.
- *-g <WxH><+X+Y>* - specifies the geometry to be used when constructing the panel. The syntax is *Width x Height + X Anchor + Y Anchor*. You can set the size or position or both. The anchor point is the upper left corner of the panel. An example is *-g 250x500+800+0* This sets the panel at 250 pixels wide, 500 pixels tall, and anchors it at X800 Y0.
- *-c panelname* - tells PyVCP what to call the component and also the title of the window. The panel-name can be any name without spaces.

To load a *stand alone* PyVCP panel without LinuxCNC use this command:

```
loadusr -Wn mypanel pyvcp -g 250x500+800+0 -c mypanel mypanel.xml
```

The minimum command to load a pyvcp panel is:

```
loadusr pyvcp mypanel.xml
```

You would use this if you want a panel without LinuxCNC's machine controller such as for testing or a standalone DRO.

The *loadusr* command is used when you also load a component that will stop HAL from closing until it's done. If you loaded a panel and then loaded Classic Ladder using *loadusr -w classicladder*, CL would hold HAL open (and the panel) until you closed CL. The *-Wn* above means wait for the component *-Wn "name"* to become ready. (*name* can be any name. Note the capital W and lowercase n.) The *-c* tells PyVCP to build a panel with the name *panelname* using the info in *panel_file_name.xml*. The name *panel_file_name.xml* can be any name but must end in *.xml* - it is the file that describes how to build the panel. You must add the path name if the panel is not in the directory that the HAL script is in.

An optional command to use if you want the panel to stop HAL from continuing commands / shutting down. After loading any other components you want the last HAL command to be:

```
waitusr panelname
```

This tells HAL to wait for component *panelname* to close before continuing HAL commands. This is usually set as the last command so that HAL shuts down when the panel is closed.

10.1.6 Widgets

HAL signals come in two variants, bits and numbers. Bits are off/on signals. Numbers can be *float*, *s32* or *u32*. For more information on HAL data types see the [HAL Data](#) section. The PyVCP widget can either display the value of the signal with an indicator widget, or modify the signal value with a control widget. Thus there are four classes of PyVCP widgets that you can connect to a HAL signal. A fifth class of helper widgets allow you to organize and label your panel.

- Widgets for indicating *bit* signals: *led*, *rectled*.
- Widgets for controlling *bit* signals: *button*, *checkboxbutton*, *radiobutton*.
- Widgets for indicating *number* signals: *number*, *s32*, *u32*, *bar*, *meter*.
- Widgets for controlling *number* signals: *spinbox*, *scale*, *jogwheel*.
- Helper widgets: *hbox*, *vbox*, *table*, *label*, *labelframe*.

10.1.6.1 Syntax

Each widget is described briefly, followed by the markup used, and a screen shot. All tags inside the main widget tag are optional.

10.1.6.2 General Notes

At the present time, both a tag-based and an attribute-based syntax are supported. For instance, the following XML fragments are treated identically:

```
<led halpin="my-led"/>
```

and

```
<led><halpin>"my-led"</halpin></led>
```

When the attribute-based syntax is used, the following rules are used to turn the attributes value into a Python value:

1. If the first character of the attribute is one of the following, it is evaluated as a Python expression:
{('"
2. If the string is accepted by `int()`, the value is treated as an integer
3. If the string is accepted by `float()`, the value is treated as floating-point
4. Otherwise, the string is accepted as a string.

When the tag-based syntax is used, the text within the tag is always evaluated as a Python expression. The examples below show a mix of formats.

Comments To add a comment use the xml syntax for a comment.

```
<!-- My Comment -->
```

Editing the XML file Edit the XML file with a text editor. In most cases you can right click on the file and select *open with text editor* or similar.

Colors

Colors can be specified using the X11 rgb colors by name *gray75* or hex *#0000ff*. A complete list is located here <http://sedition.com/perl/rgb.html>.

Common Colors (colors with numbers indicate shades of that color)

- white
- black
- blue and blue1 - 4
- cyan and cyan1 - 4
- green and green1 - 4
- yellow and yellow1 - 4
- red and red1 - 4
- purple and purple1 - 4
- gray and gray0 - 100

HAL Pins HAL pins provide a means to *connect* the widget to something. Once you create a HAL pin for your widget you can *connect* it to another HAL pin with a *net* command in a .hal file. For more information on the *net* command see the [HAL Commands](#) section.

10.1.6.3 Label

A label is a way to add text to your panel.

- `<label></label>` - creates a label
- `<text>"text"</text>` - the text to put in your label, a blank label can be used as a spacer to align other objects.
- `("Helvetica",20)` - specify the font and size of the text
- `<relief>FLAT</relief>` - specify the border around the label (*FLAT*, *RAISED*, *SUNKEN*) default is *FLAT*
- `<bd>n</bd>` - where *n* is the border width when *RAISED* or *SUNKEN* borders are used.
- `<padx>n</padx>` - where *n* is the amount of extra horizontal extra space.
- `<pady>n</pady>` - where *n* is the amount of extra vertical extra space.

The label has an optional disable pin that is created when you add `<disable_pin>True</disable_pin>`.

```
<label>
  <text>"This is a Label:"</text>
  <font>("Helvetica",20)</font>
</label>
```

The above code produced this example:



Figure 10.3: Simple Label Example

10.1.6.4 Multi_Label

An extension of the text label.

Selectable text label, can display up to 6 label legends when associated bit pin is activated.

Attach each legend pin to a signal and get a descriptive label when the signal is TRUE.

If more than one legend pin is TRUE, the highest numbered *TRUE* legend will be displayed.

If a disable pin is created with `<disable_pin>True</disable_pin>` and that pin is set to true the label changes to a grayed out state.

```
<multilabel>
  <legends>["Label1", "Label2", "Label3", "Label4", "Label5", "Label6"]</legends>
  <font>("Helvetica",20)</font>
  <disable_pin>True</disable_pin>
</multilabel>
```

The above example would create the following pins.

```
pyvcp.multilabel.0.disable
pyvcp.multilabel.0.legend0
pyvcp.multilabel.0.legend1
pyvcp.multilabel.0.legend2
pyvcp.multilabel.0.legend3
pyvcp.multilabel.0.legend4
pyvcp.multilabel.0.legend5
```

If you have more than one multilabel the pins created would increment the number like this *pyvcp.multilabel.*

10.1.6.5 LEDs

A LED is used to indicate the status of a *bit* halpin. The LED color will be *on_color* when the halpin is true, and *off_color* otherwise.

- `<led></led>` - makes a round LED
- `<rectled></rectled>` - makes a rectangle LED
- `<halpin>name</halpin>` - *name* of the pin, default is *led.n*, where *n* is an integer that is incremented for each LED.
- `<size>n</size>` - *n* is the size of the led in pixels, default is 20
- `<on_color>color</on_color>` - sets the color of the LED when the pin is true. default is *green*. See [colors](#) for more info.
- `<off_color>color</off_color>` - sets the color of the LED when the pin is false. default is *red*
- `<height>n</height>` - sets the height of the LED in pixels
- `<width>n</width>` - sets the width of the LED in pixels
- `<disable_pin>>false</disable_pin>` - when true adds a disable pin to the led.
- `<disabled_color>color</disabled_color>` - sets the color of the LED when the pin is disabled.

Round LED

```
<led>
  <halpin>"my-led"</halpin>
  <size>50</size>
  <on_color>"green"</on_color>
  <off_color>"red"</off_color>
</led>
```

The above code produced this example:

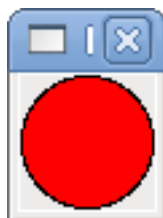


Figure 10.4: Round LED Example

Rectangle LED This is a variant of the *led* widget.

```

<vbox>
  <relief>RIDGE</relief>
  <bd>6</bd>
  <rectled>
    <halpin>"my-led"</halpin>
    <height>"50"</height>
    <width>"100"</width>
    <on_color>"green"</on_color>
    <off_color>"red"</off_color>
  </rectled>
</vbox>

```

The above code produced this example. Also showing a vertical box with relief.



Figure 10.5: Simple Rectangle LED Example

10.1.6.6 Buttons

A button is used to control a BIT pin. The pin will be set True when the button is pressed and held down, and will be set False when the button is released. Buttons can use the following optional options.

- `<padx>n</padx>` - where *n* is the amount of extra horizontal extra space.
- `<pady>n</pady>` - where *n* is the amount of extra vertical extra space.
- `<activebackground>"color"</activebackground>` - the cursor over color.
- `<fg>"color"</fg>` - the foreground color.
- `<bg>"color"</bg>` - the background color.
- `<disable_pin>True</disable_pin>` - disable pin.

Text Button A text button controls a *bit* halpin. The halpin is false until the button is pressed then it is true. The button is a momentary button.

The text button has an optional disable pin that is created when you add `<disable_pin>True</disable_pin>`.

```

<button>
  <halpin>"ok-button"</halpin>
  <text>"OK"</text>
</button>
<button>
  <halpin>"abort-button"</halpin>
  <text>"Abort"</text>
</button>

```

The above code produced this example:

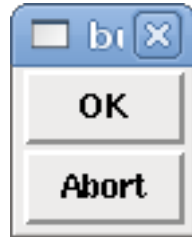


Figure 10.6: Simple Buttons Example

Checkbutton A checkbutton controls a bit halpin. The halpin will be set True when the button is checked, and false when the button is unchecked. The checkbutton is a toggle type button. The checkbuttons may be set initially as TRUE or FALSE the `initval` field A pin called `changepin` is also created automatically, which can toggle the Checkbutton via HAL, if the value linked is changed, to update the display remotely.



Figure 10.7: Unchecked button



Figure 10.8: Checked button

Checkbutton Code Example

```
<checkbutton>
  <halpin>"coolant-chkbtn"</halpin>
  <text>"Coolant"</text>
  <initval>1</initval>
</checkbutton>
<checkbutton>
  <halpin>"chip-chkbtn"</halpin>
  <text>"Chips  "</text>
  <initval>0</initval>
</checkbutton>
```

The above code produced this example:

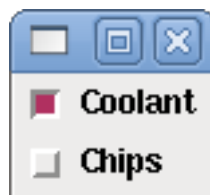


Figure 10.9: Simple Checkbutton Example

The coolant checkbox is checked.

Notice the extra spaces in the "Chips" text to keep the checkboxes aligned.

Radiobutton A radiobutton will set one of the halpins true. The other pins are set false. The initval field may be set to choose the default selection when the panel displays. Only one radio button may be set to TRUE (1) or only the highest number pin set TRUE will have that value.

```
<radiobutton>
  <choices>["one","two","three"]</choices>
  <halpin>"my-radio"</halpin>
  <initval>0</initval>
</radiobutton>
```

The above code produced this example:

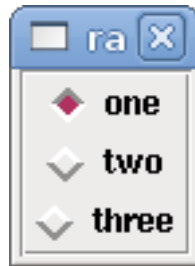


Figure 10.10: Simple Radiobutton Example

Note that the HAL pins in the example above will be named my-radio.one, my-radio.two, and my-radio.three.

In the image above, *one* is the selected value.

Use this tag `<orient>HORIZONTAL</orient>` to display horizontally.

10.1.6.7 Number Displays

Number displays can use the following formatting options

- `("Font Name",n)` where *n* is the font size
- `<width>n</width>` where *n* is the overall width of the space used
- `<justify>pos</justify>` where *pos* is LEFT, CENTER, or RIGHT (doesn't work)
- `<padx>n</padx>` where *n* is the amount of extra horizontal extra space
- `<pady>n</pady>` where *n* is the amount of extra vertical extra space

Number The number widget displays the value of a float signal.

```
<number>
  <halpin>"my-number"</halpin>
  <font>("Helvetica",24)</font>
  <format>" +4.4f"</format>
</number>
```

The above code produced this example:

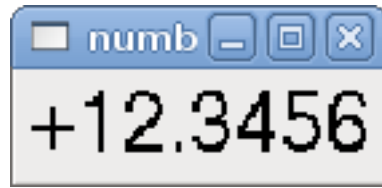


Figure 10.11: Simple Number Example

- `` - is a Tkinter font type and size specification. One font that will show up to at least size 200 is *courier 10 pitch*, so for a really big Number widget you could specify:

```
<font>("courier 10 pitch",100)</font>
```

- `<format>` - is a C-style format specified that determines how the number is displayed.

s32 Number The s32 number widget displays the value of a s32 number. The syntax is the same as *number* except the name which is `<s32>`. Make sure the width is wide enough to cover the largest number you expect to use.

```
<s32>
  <halpin>"my-number"</halpin>
  <font>("Helvetica",24)</font>
  <format>"6d"</format>
  <width>6</width>
</s32>
```

The above code produced this example:

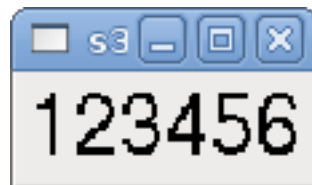


Figure 10.12: Simple s32 Number Example

u32 Number The u32 number widget displays the value of a u32 number. The syntax is the same as *number* except the name which is `<u32>`.

Bar A bar widget displays the value of a FLOAT signal both graphically using a bar display and numerically. The color of the bar can be set as one color throughout its range (default using fillcolor) or set to change color dependent upon the value of the halpin (range1, range2 range3 must all be set, if you only want 2 ranges, set 2 of them to the same color).

- `<halpin>"my-bar"</halpin>` text, sets the pin name, pyvcp.my-bar
- `<min_>0</min_>` number, sets the minimum scale
- `<max_>140</max_>` number, sets the maximum scale
- `<format>"3.1f"</format>` text, sets the number format using python number formatting
- `<bgcolor>"grey"</bgcolor>` text, sets the background color

- `<fillcolor>"red"</fillcolor>` text, sets the fill color
- `<range1>0,100,"green"</range1>` number, number, text, sets the first range and color
- `<range2>101,135,"orange"</range2>` number, number, text, sets the first range and color
- `<range3>136, 150,"red"</range3>` number, number, text, sets the first range and color
- `<canvas_width>200</canvas_width>` number, sets the overall width
- `<canvas_height>50</canvas_height>` number, sets the overall height
- `<bar_height>30</bar_height>` number, sets the bar height, must be less than `canvas_height`
- `<bar_width>150</bar_width>` number, sets the bar width, must be less than `canvas_width`

```
<bar>
  <halpin>"my-bar"</halpin>
  <min_>0</min_>
  <max_>123</max_>
  <format>"3.1f"</format>
  <bgcolor>"grey"</bgcolor>
  <fillcolor>"red"</fillcolor>
  <range1>0,100,"green"</range1>
  <range2>101,135,"orange"</range2>
  <range3>136, 150,"red"</range3>
  <canvas_width>200</canvas_width>
  <canvas_height>50</canvas_height>
  <bar_height>30</bar_height>
  <bar_width>150</bar_width>
</bar>
```

The above code produced this example:

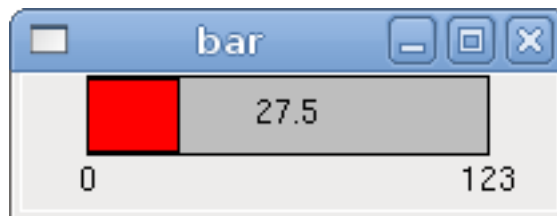


Figure 10.13: Simple Bar Example

Meter Meter displays the value of a FLOAT signal using a traditional dial indicator.

```
<meter>
  <halpin>"mymeter"</halpin>
  <text>"Battery"</text>
  <subtext>"Volts"</subtext>
  <size>250</size>
  <min_>0</min_>
  <max_>15.5</max_>
  <majorscale>1</majorscale>
  <minorscale>0.2</minorscale>
  <region1>(14.5,15.5,"yellow")</region1>
  <region2>(12,14.5,"green")</region2>
  <region3>(0,12,"red")</region3>
</meter>
```


The above code produced this example:

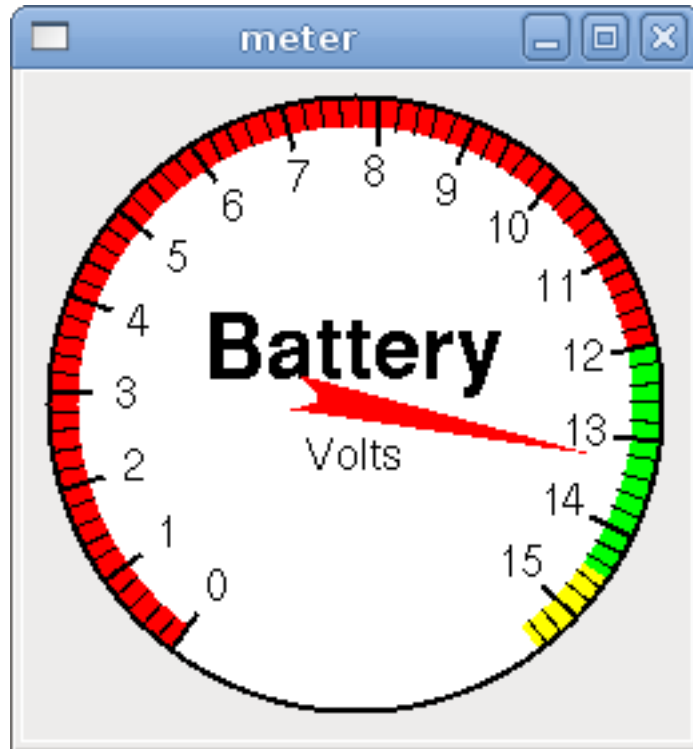


Figure 10.14: Simple Meter Example

10.1.6.8 Number Inputs

Spinbox Spinbox controls a FLOAT pin. You increase or decrease the value of the pin by either pressing on the arrows, or pointing at the spinbox and rolling your mouse-wheel. If the param_pin field is set TRUE(1), a pin will be created that can be used to set the spinbox to an initial value and to remotely alter its value without HID input.

```
<spinbox>
  <halpin>"my-spinbox"</halpin>
  <min_>-12</min_>
  <max_>33</max_>
  <initval>0</initval>
  <resolution>0.1</resolution>
  <format>"2.3f"</format>
  <font>("Arial",30)</font>
  <param_pin>1</param_pin>
</spinbox>
```

The above code produced this example:



Figure 10.15: Simple Spinbox Example

Scale Scale controls a float or a s32 pin. You increase or decrease the value of the pin by either dragging the slider, or pointing at the scale and rolling your mouse-wheel. The *halpin* will have both *-f* and *-i* added to it to form the float and s32 pins. Width is the width of the slider in vertical and the height of the slider in horizontal orientation. If the *param_pin* field is set TRUE(1), a pin will be created that can be used to set the spinbox to an initial value and to remotely alter its value without HID input.

```
<scale>
  <font>("Helvetica",16)</font>
  <width>"25"</width>
  <halpin>"my-hscale"</halpin>
  <resolution>0.1</resolution>
  <orient>HORIZONTAL</orient>
  <initval>-15</initval>
  <min_>-33</min_>
  <max_>26</max_>
  <param_pin>1</param_pin>
</scale>
<scale>
  <font>("Helvetica",16)</font>
  <width>"50"</width>
  <halpin>"my-vscales"</halpin>
  <resolution>1</resolution>
  <orient>VERTICAL</orient>
  <min_>100</min_>
  <max_>0</max_>
  <param_pin>1</param_pin>
</scale>
```

The above code produced this example:



Figure 10.16: Simple Scale Example

Note

Note that by default it is "min" which is displayed even if it is greater than "max", unless "min" is negative.

Dial The Dial outputs a HAL float and reacts to both mouse wheel and dragging. Double left click to increase the resolution and double right click to reduce the resolution by one digit. The output is capped by the min and max values. The `<cpr>` is how many tick marks are on the outside of the ring (beware of high numbers). If the `param_pin` field is set TRUE(1), a pin will be created that can be used to set the spinbox to an initial value and to remotely alter its value without HID input.

```
<dial>
  <size>200</size>
  <cpr>100</cpr>
  <min_>-15</min_>
  <max_>15</max_>
  <text>"Dial"</text>
  <initval>0</initval>
  <resolution>0.001</resolution>
  <halpin>"anaout"</halpin>
  <dialcolor>"yellow"</dialcolor>
  <edgecolor>"green"</edgecolor>
  <dotcolor>"black"</dotcolor>
  <param_pin>1</param_pin>
</dial>
```

The above code produced this example:

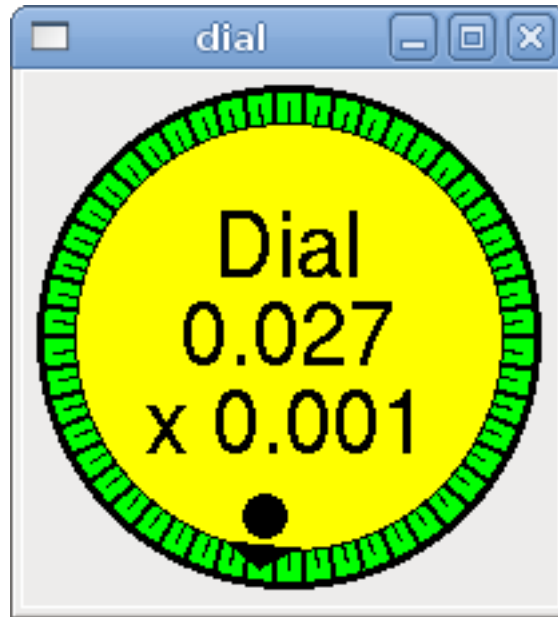


Figure 10.17: Simple Dial Example

Jogwheel Jogwheel mimics a real jogwheel by outputting a FLOAT pin which counts up or down as the wheel is turned, either by dragging in a circular motion, or by rolling the mouse-wheel.

Optional tags: * `<text>"My Text"</text>` displays text * `<bgcolor>"grey"</bgcolor>` `<fillcolor>"green"</fillcolor>` background & active colors * `<scale_pin>1</scale_pin>` creates scale text and a FLOAT.scale pin to display jog scale * `<clear_pin>1</clear_pin>` creates DRO and a BIT.reset pin to reset DRO. Needs scale_pin for scaled DRO. shift+click resets DRO also

```
<jogwheel>
  <halpin>"my-wheel"</halpin>
  <cpr>45</cpr>
  <size>250</size>
</jogwheel>
```

The above code produced this example:



Figure 10.18: Simple Jogwheel Example

10.1.6.9 Images

Image displays use only .gif image format. All of the images must be the same size. The images must be in the same directory as your ini file (or in the current directory if running from the command line with halrun/halcmd).

Image Bit The *image_bit* toggles between two images by setting the halpin to true or false.

```
<image name='fwd' file='fwd.gif' />
<image name='rev' file='rev.gif' />
<vbox>
  <image_bit halpin='selectimage' images='fwd rev' />
</vbox>
```

This example was produced from the above code. Using the two image files fwd.gif and rev.gif. FWD is displayed when *selectimage* is false and REV is displayed when *selectimage* is true.



Figure 10.19: Selectimage False Example



Figure 10.20: Selectimage True Example

Image u32 The *image_u32* is the same as *image_bit* except you have essentially an unlimited number of images and you *select* the image by setting the halpin to a integer value with 0 for the first image in the images list and 1 for the second image etc.

```
<image name='stb' file='stb.gif'/>
<image name='fwd' file='fwd.gif'/>
<image name='rev' file='rev.gif'/>
<vbox>
  <image_u32 halpin='selectimage' images='stb fwd rev'/>
</vbox>
```

The above code produced the following example by adding the stb.gif image.



Figure 10.21: Simple image_u32 Example with halpin=0

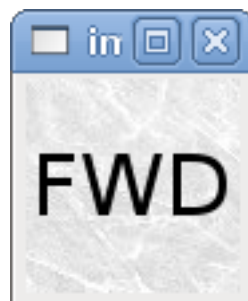


Figure 10.22: Simple image_u32 Example with halpin=1



Figure 10.23: Simple image_u32 Example withhalpin=2

Notice that the default is the min even though it is set higher than max unless there is a negative min.

10.1.6.10 Containers

Containers are widgets that contain other widgets. Containers are used to group other widgets.

Borders Container borders are specified with two tags used together. The `<relief>` tag specifies the type of border and the `<bd>` specifies the width of the border.

`<relief>type</relief>`

Where *type* is FLAT, SUNKEN, RAISED, GROOVE, or RIDGE

`<bd>n</bd>`

Where *n* is the width of the border.

```
<hbox>
  <button>
    <relief>FLAT</relief>
    <text>"FLAT"</text>
    <bd>3</bd>
  </button>
  <button>
    <relief>SUNKEN</relief>
    <text>"SUNKEN"</text>
    <bd>3</bd>
  </button>
  <button>
    <relief>RAISED</relief>
    <text>"RAISED"</text>
    <bd>3</bd>
  </button>
  <button>
    <relief>GROOVE</relief>
    <text>"GROOVE"</text>
    <bd>3</bd>
  </button>
  <button>
    <relief>RIDGE</relief>
    <text>"RIDGE"</text>
    <bd>3</bd>
  </button>
</hbox>
```

The above code produced this example:



Figure 10.24: Containers Borders Showcase

Fill Container fill are specified with the `<boxfill fill=""/>` tag. Valid entries are none, x, y and both. The x fill is a horizontal fill and the y fill is a vertical fill

`<boxfill fill = "style"/>`

Where *style* is none, x, y, or both. Default is x for VBox and y for Hbox.

Anchor Container anchors are specified with the `<boxanchor anchor=""/>` tag. The anchor specifies where to position each slave in its parcel. Valid entries are center, n, s, e, w, for center, north, south, east and west. Combinations like sw, se, nw and ne are also valid.

`<boxanchor anchor="position"/>`

Where *position* is center, n, s, e, w, ne, nw, se or sw. Default is center.

Expand Container expand is specified with the boolean `<boxexpand expand=""/>` tag. Valid entries are yes, no.

`<boxexpand expand="boolean"/>`

Where *boolean* is either yes or no. Default is yes.

Hbox Use an Hbox when you want to stack widgets horizontally next to each other.

```
<hbox>
  <relief>RIDGE</relief>
  <bd>6</bd>
  <label><text>"a hbox:"</text></label>
  <led></led>
  <number></number>
  <bar></bar>
</hbox>
```

The above code produced this example:

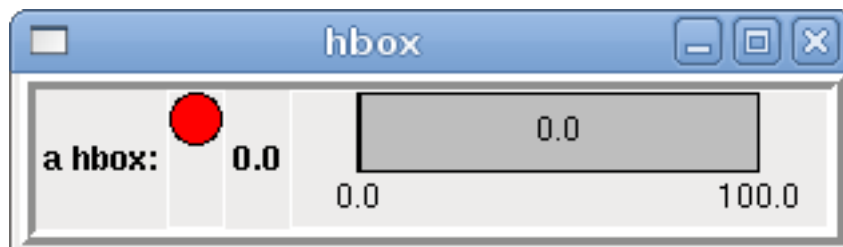


Figure 10.25: Simple hbox Example

Inside an Hbox, you can use the `<boxfill fill=""/>`, `<boxanchor anchor=""/>`, and `<boxexpand expand=""/>` tags to choose how items in the box behave when the window is re-sized. The default is `fill="y"`, `anchor="center"`, `expand="yes"` for a Hbox.

Vbox Use a VBox when you want to stack widgets vertically on top of each other.


```

<vbox>
  <relief>RIDGE</relief>
  <bd>6</bd>
  <label><text>"a vbox:"</text></label>
  <led></led>
  <number></number>
  <bar></bar>
</vbox>

```

The above code produced this example:



Figure 10.26: Simple vbox Example

Inside a VBox, you can use the `<boxfill fill=""/>`, `<boxanchor anchor=""/>`, and `<boxexpand expand=""/>` tags to choose how items in the box behave when the window is re-sized. The default is `fill="x"`, `anchor="center"`, `expand="yes"` for a Hbox.

Labelframe A labelframe is a frame with a groove and a label at the upper-left corner.

```

<labelframe text="Label: Leds groupées">

```

```

  <labelframe text="Group Title">
    <font>("Helvetica",16)</font>
    <hbox>
      <led/>
      <led/>
    </hbox>
  </labelframe>
</labelframe>

```

The above code produced this example:



Figure 10.27: Simple labelframe Example

Table A table is a container that allows layout in a grid of rows and columns. Each row is started by a `<tablerow/>` tag. A contained widget may span rows or columns through the use of the `<tablespan`

`rows= cols=` tag. The sides of the cells to which the contained widgets "stick" may be set through the use of the `<tablesticky sticky=` tag. A table expands on its flexible rows and columns.

Table Code Example

```
<table flexible_rows="[2]" flexible_columns="[1,4]">
<tablesticky sticky="new"/>
<tablerow/>
  <label>
    <text>" A (cell 1,1) "</text>
    <relief>RIDGE</relief>
    <bd>3</bd>
  </label>
  <label text="B (cell 1,2)"/>
  <tablespan columns="2"/>
  <label text="C, D (cells 1,3 and 1,4)"/>
</tablerow/>
  <label text="E (cell 2,1)"/>
  <tablesticky sticky="nsew"/>
  <tablespan rows="2"/>
  <label text="'spans\n2 rows'"/>
  <tablesticky sticky="new"/>
  <label text="G (cell 2,3)"/>
  <label text="H (cell 2,4)"/>
</tablerow/>
  <label text="J (cell 3,1)"/>
  <label text="K (cell 3,2)"/>
  <u32 halpin="test"/>
</table>
```

The above code produced this example:

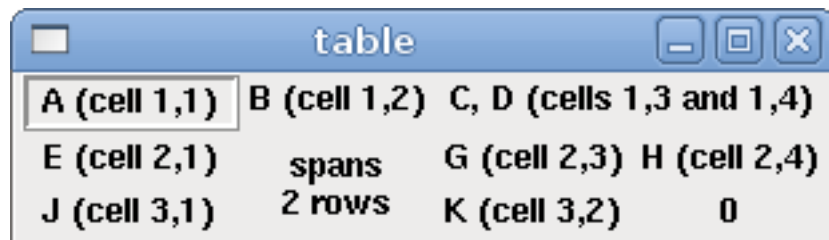


Figure 10.28: Table Example

Tabs A tabbed interface can save quite a bit of space.

```
<tabs>
  <names> ["spindle", "green eggs"]</names>
</tabs>
<tabs>
  <names>["Spindle", "Green Eggs", "Ham"]</names>
  <vbox>
    <label>
      <text>"Spindle speed:"</text>
    </label>
    <bar>
      <halpin>"spindle-speed"</halpin>
      <max_>5000</max_>
    </bar>
  </vbox>
</vbox>
```

```
<label>
  <text>"(this is the green eggs tab)"</text>
</label>
</vbox>
<vbox>
  <label>
    <text>"(this tab has nothing on it)"</text>
  </label>
</vbox>
</tabs>
```

The above code produced this example showing each tab selected.





Figure 10.29: Simple Tabs Example

10.2 PyVCP Examples

10.2.1 AXIS

To create a PyVCP panel to use with the AXIS interface that is attached to the right of AXIS you need to do the following basic things.

- Create an .xml file that contains your panel description and put it in your config directory.
- Add the PyVCP entry to the [DISPLAY] section of the ini file with your .xml file name.
- Add the POSTGUI_HALFILE entry to the [HAL] section of the ini file with the name of your postgui HAL file name.
- Add the links to HAL pins for your panel in the postgui.hal file to *connect* your PyVCP panel to LinuxCNC.

10.2.2 Floating Panels

To create floating PyVCP panels that can be used with any interface you need to do the following basic things.

- Create an .xml file that contains your panel description and put it in your config directory.
- Add a loadusr line to your .hal file to load each panel.
- Add the links to HAL pins for your panel in the postgui.hal file to *connect* your PyVCP panel to LinuxCNC.

The following is an example of a loadusr command to load two PyVCP panels and name each one so the connection names in HAL will be known.

```
loadusr -Wn btnpanel pyvcp -c btnpanel panel1.xml
loadusr -Wn spanel pyvcp -c spanel panel2.xml
```

The `-Wn` makes HAL *Wait for name* to be loaded before proceeding.

The `pyvcp -c` makes PyVCP name the panel.

The HAL pins from `panel1.xml` will be named `btnpanel.<pin name>`

The HAL pins from `panel2.xml` will be named `sppanel.<pin name>`

Make sure the `loadusr` line is before any nets that make use of the PyVCP pins.

10.2.3 Jog Buttons Example

In this example we will create a PyVCP panel with jog buttons for X, Y, and Z. This configuration will be built upon a Stepconf Wizard generated configuration. First we run the Stepconf Wizard and configure our machine, then on the Advanced Configuration Options page we make a couple of selections to add a blank PyVCP panel as shown in the following figure. For this example we named the configuration `pyvcp_xyz` on the Basic Machine Information page of the Stepconf Wizard.



Figure 10.30: XYZ Wizard Configuration

The Stepconf Wizard will create several files and place them in the `linuxcnc/configs/pyvcp_xyz` directory. If you left the create link checked you will have a link to those files on your desktop.

10.2.3.1 Create the Widgets

Open up the `custompanel.xml` file by right clicking on it and selecting *open with text editor*. Between the `<pyvcp>``</pyvcp>` tags we will add the widgets for our panel.

Look in the PyVCP Widgets Reference section of the manual for more detailed information on each widget [documentation des widgets](#).

In your custompanel.xml file we will add the description of the widgets.

```
<pyvcp>
  <labelframe text="Jog Buttons">
    <font>("Helvetica",16)</font>

    <!-- the X jog buttons -->
    <hbox>
      <relief>RAISED</relief>
      <bd>3</bd>
      <button>
        <font>("Helvetica",20)</font>
        <width>3</width>
        <halpin>"x-plus"</halpin>
        <text>"X+"</text>
      </button>
      <button>
        <font>("Helvetica",20)</font>
        <width>3</width>
        <halpin>"x-minus"</halpin>
        <text>"X- "</text>
      </button>
    </hbox>

    <!-- the Y jog buttons -->
    <hbox>
      <relief>RAISED</relief>
      <bd>3</bd>
      <button>
        <font>("Helvetica",20)</font>
        <width>3</width>
        <halpin>"y-plus"</halpin>
        <text>"Y+"</text>
      </button>
      <button>
        <font>("Helvetica",20)</font>
        <width>3</width>
        <halpin>"y-minus"</halpin>
        <text>"Y- "</text>
      </button>
    </hbox>

    <!-- the Z jog buttons -->
    <hbox>
      <relief>RAISED</relief>
      <bd>3</bd>
      <button>
        <font>("Helvetica",20)</font>
        <width>3</width>
        <halpin>"z-plus"</halpin>
        <text>"Z+"</text>
      </button>
      <button>
        <font>("Helvetica",20)</font>
        <width>3</width>
        <halpin>"z-minus"</halpin>
        <text>"Z- "</text>
      </button>
    </hbox>

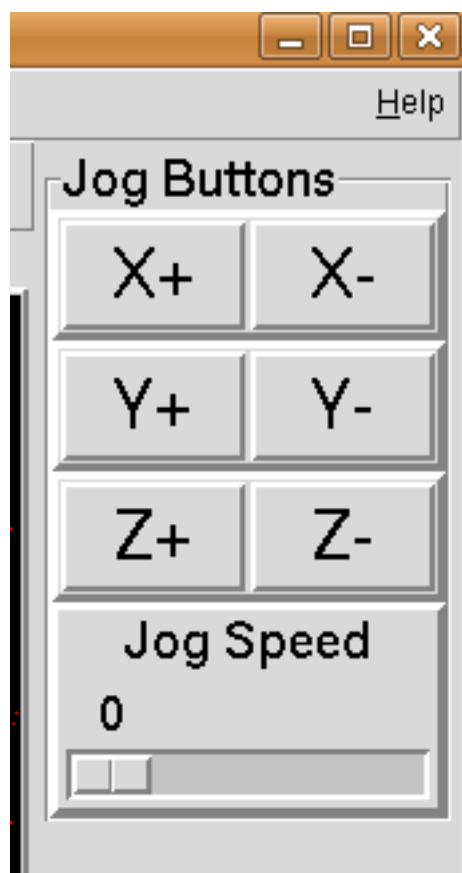
    <!-- the jog speed slider -->
    <vbox>
```

```

<relief>RAISED</relief>
<bd>3</bd>
<label>
  <text>"Jog Speed"</text>
  <font>("Helvetica",16)</font>
</label>
<scale>
  <font>("Helvetica",14)</font>
  <halpin>"jog-speed"</halpin>
  <resolution>1</resolution>
  <orient>HORIZONTAL</orient>
  <min_>0</min_>
  <max_>80</max_>
</scale>
</vbox>
</labelframe>
</pyvcp>

```

After adding the above you now will have a PyVCP panel that looks like the following attached to the right side of AXIS. It looks nice but it does not do anything until you *connect* the buttons to halui. If you get an error when you try and run scroll down to the bottom of the pop up window and usually the error is a spelling or syntax error and it will be there.



10.2.3.2 Make Connections

To make the connections needed open up your custom_postgui.hal file and add the following.

```

# connect the X PyVCP buttons
net my-jogxminus halui.axis.x.minus <= pyvcp.x-minus

```



```

net my-jogxplus halui.axis.x.plus <= pyvcp.x-plus

# connect the Y PyVCP buttons
net my-jogyminus halui.axis.y.minus <= pyvcp.y-minus
net my-jogyplus halui.axis.y.plus <= pyvcp.y-plus

# connect the Z PyVCP buttons
net my-jogzminus halui.axis.z.minus <= pyvcp.z-minus
net my-jogzplus halui.axis.z.plus <= pyvcp.z-plus

# connect the PyVCP jog speed slider
net my-jogspeed halui.axis.jog-speed <= pyvcp.jog-speed-f

```

After resetting the E-Stop and putting it into jog mode and moving the jog speed slider in the PyVCP panel to a value greater than zero the PyVCP jog buttons should work. You can not jog when running a G-code file or while paused or while the MDI tab is selected.

10.2.4 Port Tester

This example shows you how to make a simple parallel port tester using PyVCP and HAL.

First create the ptest.xml file with the following code to create the panel description.

```

<!-- Test panel for the parallel port cfg for out -->
<pyvcp>
  <hbox>
    <relief>RIDGE</relief>
    <bd>2</bd>
    <button>
      <halpin>"btn01"</halpin>
      <text>"Pin 01"</text>
    </button>
    <led>
      <halpin>"led-01"</halpin>
      <size>25</size>
      <on_color>"green"</on_color>
      <off_color>"red"</off_color>
    </led>
  </hbox>
  <hbox>
    <relief>RIDGE</relief>
    <bd>2</bd>
    <button>
      <halpin>"btn02"</halpin>
      <text>"Pin 02"</text>
    </button>
    <led>
      <halpin>"led-02"</halpin>
      <size>25</size>
      <on_color>"green"</on_color>
      <off_color>"red"</off_color>
    </led>
  </hbox>
  <hbox>
    <relief>RIDGE</relief>
    <bd>2</bd>
    <label>
      <text>"Pin 10"</text>
      <font>("Helvetica",14)</font>
    </label>

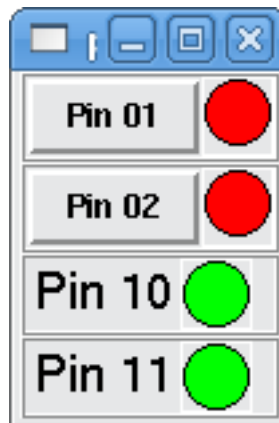
```

```

    <led>
      <halpin>"led-10"</halpin>
      <size>25</size>
      <on_color>"green"</on_color>
      <off_color>"red"</off_color>
    </led>
  </hbox>
  <hbox>
    <relief>RIDGE</relief>
    <bd>2</bd>
    <label>
      <text>"Pin 11"</text>
      <font>("Helvetica",14)</font>
    </label>
    <led>
      <halpin>"led-11"</halpin>
      <size>25</size>
      <on_color>"green"</on_color>
      <off_color>"red"</off_color>
    </led>
  </hbox>
</pyvcp>

```

This will create the following floating panel which contains a couple of in pins and a couple of out pins.



To run the HAL commands that we need to get everything up and running we put the following in our ptest.hal file.

```

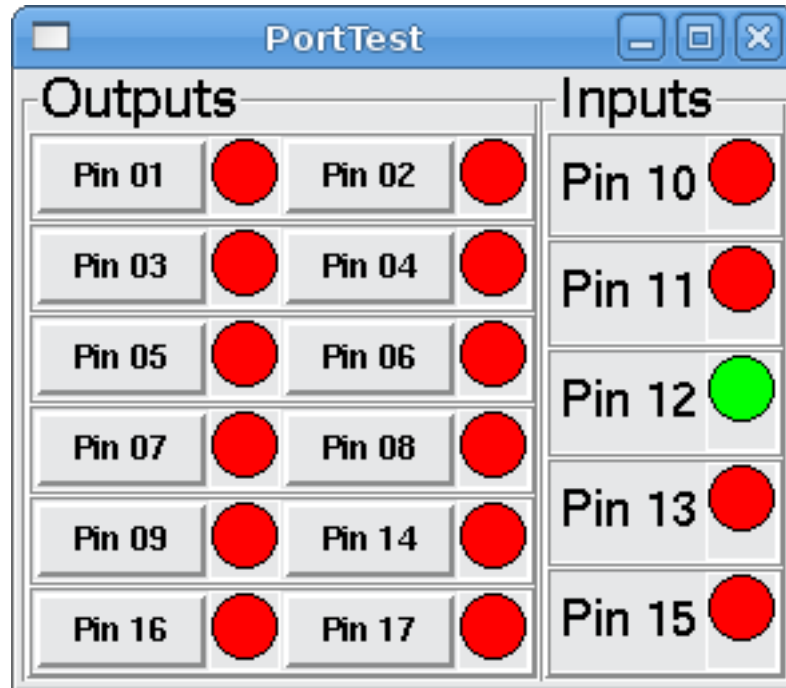
loadrt hal_parport cfg="0x378 out"
loadusr -Wn ptest pyvcp -c ptest ptest.xml
loadrt threads namel=porttest period1=1000000
addf parport.0.read porttest
addf parport.0.write porttest
net pin01 ptest.btn01 parport.0.pin-01-out ptest.led-01
net pin02 ptest.btn02 parport.0.pin-02-out ptest.led-02
net pin10 parport.0.pin-10-in ptest.led-10
net pin11 parport.0.pin-11-in ptest.led-11
start

```

To run the HAL file we use the following command from a terminal window.

```
~$ halrun -I -f ptest.hal
```

The following figure shows what a complete panel might look like.



To add the rest of the parallel port pins just modify the .xml and .hal files.

To show the pins after running the HAL script use the following command at the halcmd prompt:

```
halcmd: show pin
Component Pins:
Owner Type Dir Value Name
2 bit IN FALSE parport.0.pin-01-out <== pin01
2 bit IN FALSE parport.0.pin-02-out <== pin02
2 bit IN FALSE parport.0.pin-03-out
2 bit IN FALSE parport.0.pin-04-out
2 bit IN FALSE parport.0.pin-05-out
2 bit IN FALSE parport.0.pin-06-out
2 bit IN FALSE parport.0.pin-07-out
2 bit IN FALSE parport.0.pin-08-out
2 bit IN FALSE parport.0.pin-09-out
2 bit OUT TRUE parport.0.pin-10-in ==> pin10
2 bit OUT FALSE parport.0.pin-10-in-not
2 bit OUT TRUE parport.0.pin-11-in ==> pin11
2 bit OUT FALSE parport.0.pin-11-in-not
2 bit OUT TRUE parport.0.pin-12-in
2 bit OUT FALSE parport.0.pin-12-in-not
2 bit OUT TRUE parport.0.pin-13-in
2 bit OUT FALSE parport.0.pin-13-in-not
2 bit IN FALSE parport.0.pin-14-out
2 bit OUT TRUE parport.0.pin-15-in
2 bit OUT FALSE parport.0.pin-15-in-not
2 bit IN FALSE parport.0.pin-16-out
2 bit IN FALSE parport.0.pin-17-out
4 bit OUT FALSE ptest.btn01 ==> pin01
4 bit OUT FALSE ptest.btn02 ==> pin02
4 bit IN FALSE ptest.led-01 <== pin01
4 bit IN FALSE ptest.led-02 <== pin02
4 bit IN TRUE ptest.led-10 <== pin10
4 bit IN TRUE ptest.led-11 <== pin11
```

This will show you what pins are IN and what pins are OUT as well as any connections.

10.2.5 GS2 RPM Meter

The following example uses the Automation Direct GS2 VDF driver and displays the RPM and other info in a PyVCP panel. This example is based on the GS2 example in the Hardware Examples section this manual.

10.2.5.1 The Panel

To create the panel we add the following to the .xml file.

```
<pyvcp>

  <!-- the RPM meter -->
  <hbox>
    <relief>RAISED</relief>
    <bd>3</bd>
    <meter>
      <halpin>"spindle_rpm"</halpin>
      <text>"Spindle"</text>
      <subtext>"RPM"</subtext>
      <size>200</size>
      <min_>0</min_>
      <max_>3000</max_>
      <majorscale>500</majorscale>
      <minorscale>100</minorscale>
      <region1>0,10,"yellow"</region1>
    </meter>
  </hbox>

  <!-- the On Led -->
  <hbox>
    <relief>RAISED</relief>
    <bd>3</bd>
    <vbox>
      <relief>RAISED</relief>
      <bd>2</bd>
      <label>
        <text>"On"</text>
        <font>("Helvetica",18)</font>
      </label>
      <width>5</width>
      <hbox>
        <label width="2"/> <!-- used to center the led -->
        <rectled>
          <halpin>"on-led"</halpin>
          <height>"30"</height>
          <width>"30"</width>
          <on_color>"green"</on_color>
          <off_color>"red"</off_color>
        </rectled>
      </hbox>
    </vbox>

    <!-- the FWD Led -->
    <vbox>
      <relief>RAISED</relief>
      <bd>2</bd>
      <label>
        <text>"FWD"</text>
        <font>("Helvetica",18)</font>
```

```

    <width>5</width>
  </label>
<label width="2"/>
  <rectled>
    <halpin>"fwd-led"</halpin>
    <height>"30"</height>
    <width>"30"</width>
    <on_color>"green"</on_color>
    <off_color>"red"</off_color>
  </rectled>
</vbox>

<!-- the REV Led -->
<vbox>
<relief>RAISED</relief>
<bd>2</bd>
  <label>
    <text>"REV"</text>
    <font>("Helvetica",18)</font>
    <width>5</width>
  </label>
<label width="2"/>
  <rectled>
    <halpin>"rev-led"</halpin>
    <height>"30"</height>
    <width>"30"</width>
    <on_color>"red"</on_color>
    <off_color>"green"</off_color>
  </rectled>
</vbox>
</hbox>
</pyvcp>

```

The above gives us a PyVCP panel that looks like the following.



10.2.5.2 The Connections

To make it work we add the following code to the custom_postgui.hal file.

```
# display the rpm based on freq * rpm per hz
loadrt mult2
addf mult2.0 servo-thread
setp mult2.0.in1 28.75
net cypher_speed mult2.0.in0 <= spindle-vfd.frequency-out
net speed_out pyvcp.spindle_rpm <= mult2.0.out

# run led
net gs2-run => pyvcp.on-led

# fwd led
net gs2-fwd => pyvcp.fwd-led

# rev led
net running-rev spindle-vfd.spindle-rev => pyvcp.rev-led
```

Some of the lines might need some explanations. The fwd led line uses the signal created in the custom.hal file whereas the rev led needs to use the spindle-rev bit. You can't link the spindle-fwd bit twice so you use the signal that it was linked to.

10.2.6 Rapid to Home Button

This example creates a button on the PyVCP side panel when pressed will send all the axis back to the home position. This example assumes you don't have a PyVCP panel.



In your configuration directory create the .xml file. In this example it's named *rth.xml*. In the *rth.xml* file add the following code to create the button.

```
<pyvcp>
<!-- rapid to home button example -->
<button>
<halpin>"rth-button"</halpin>
<text>"Rapid to Home"</text>
</button>
</pyvcp>
```

Open your .ini file with a text editor and in the [DISPLAY] section add the following line. This is what loads the PyVCP panel.

```
PYVCP = rth.xml
```

If you don't have a [HALUI] section in the ini file create it and add the following MDI command.

```
MDI_COMMAND = G53 G0 X0 Y0 Z0
```

Note

Information about [G53](#) and [G0](#) G-codes

In the [HAL] section if you don't have a post gui file add the following and create a file called *postgui.hal*.

```
POSTGUI_HALFILE = postgui.hal
```

In the *postgui.hal* file add the following code to link the PyVCP button to the MDI command.

```
net rth halui.mdi-command-00 <= pyvcp.rth-button
```

Note

Information about the [net](#) command

10.3 GladeVCP: Glade Virtual Control Panel

10.3.1 What is GladeVCP?

GladeVCP is a LinuxCNC component which adds the ability to add a new user interface panel to LinuxCNC user interfaces like:

- AXIS
- Touchy
- Gscreen
- Gmoccapy

Unlike PyVCP, GladeVCP is not limited to displaying and setting HAL pins, as arbitrary actions can be executed in Python code - in fact, a complete LinuxCNC user interface could be built with GladeVCP and Python.

GladeVCP uses the [Glade](#) WYSIWYG user interface editor, which makes it easy to create visually pleasing panels. It relies on the [PyGObject](#) bindings to the rich [GTK+](#) widget set, and in fact all of these may be used in a GladeVCP application - not just the specialized widgets for interacting with HAL and LinuxCNC, which are documented here.

10.3.1.1 PyVCP versus GladeVCP at a glance

Both support the creation of panels with *HAL widgets* - user interface elements like LED's, buttons, sliders etc whose values are linked to a HAL pin, which in turn interfaces to the rest of LinuxCNC.

PyVCP:

- Widget set: uses TkInter widgets.
- User interface creation: "edit XML file / run result / evaluate looks" cycle.
- No support for embedding user-defined event handling.
- No LinuxCNC interaction beyond HAL pin I/O supported.

GladeVCP:

- Widget set: relies on the [GTK+](#) widget set.
-

- User interface creation: uses the [Glade](#) WYSIWYG user interface editor.
- Any HAL pin change may be directed to call back into a user-defined Python event handler
- Any GTK signal (key/button press, window, I/O, timer, network events) may be associated with user-defined handlers in Python.
- Direct LinuxCNC interaction: arbitrary command execution, like initiating MDI commands to call a G-code subroutine, plus support for status change operations through Action Widgets.
- Several independent GladeVCP panels may be run in different tabs.
- Separation of user interface appearance and functionality: change appearance without touching any code.

10.3.2 A Quick Tour with the Example Panel

GladeVCP panel windows may be run in three different setups:

- always visible integrated into AXIS at the right side, exactly like PyVCP panels,
- as a tab in AXIS, Touchy, Gscreen, or Gmoccapy; in AXIS this would create a third tab besides the Preview and DRO tabs which must be raised explicitly,
- as a standalone toplevel window, which can be iconified/deiconified independent of the main window.

Installed LinuxCNC If you're using an installed version of LinuxCNC the examples shown below are in the [configuration picker](#) in the *Sample Configurations > apps > GladeVCP* branch.

Git Checkout The following instructions only apply if you're using a git checkout. Open a terminal and change to the directory created by git then issue the commands as shown.

Note

For the following commands to work on your git checkout you must first run *make* then run *sudo make setuid* then run *./scripts/rip-environment*. More information about a git checkout is on the LinuxCNC wiki page.

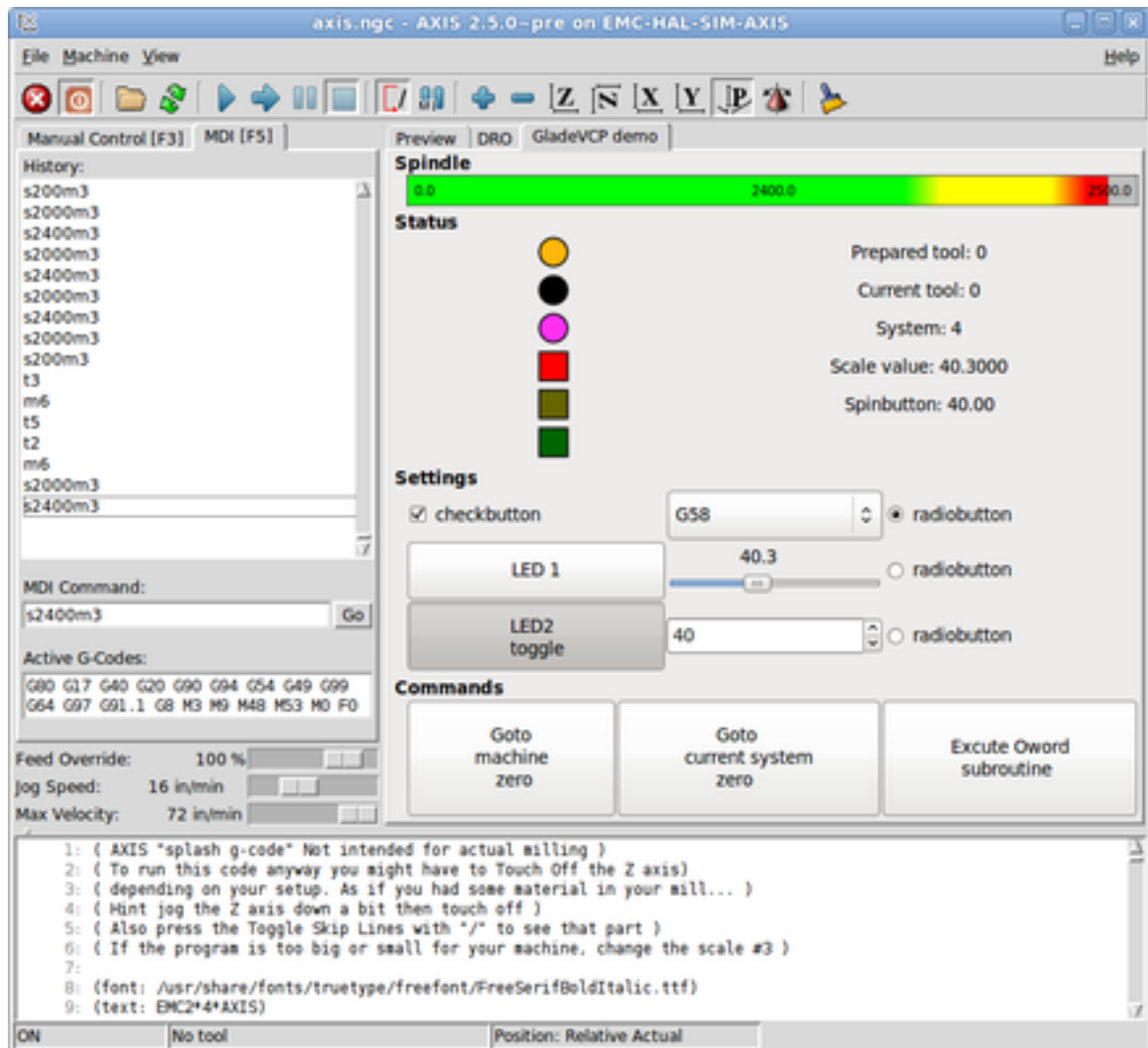
Run the sample GladeVCP panel integrated into AXIS like PyVCP as follows:

```
$ cd configs/sim/axis/gladevcp
$ linuxcnc gladevcp_panel.ini
```



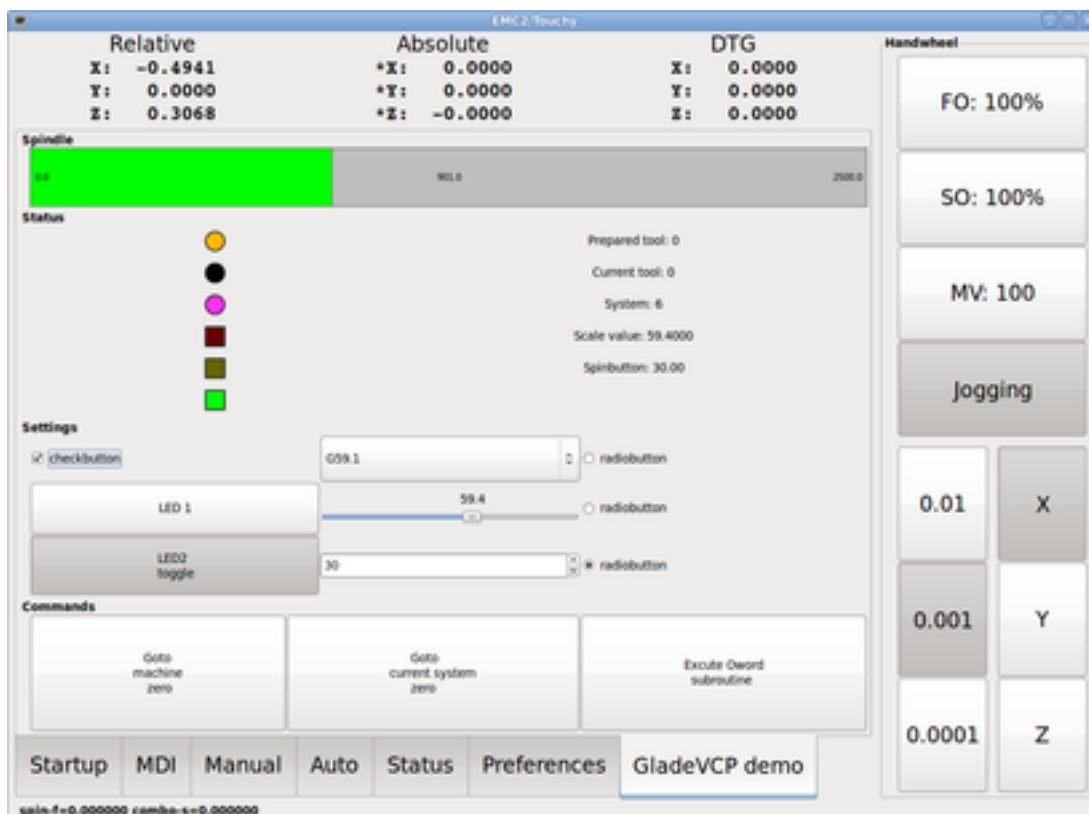
Run the same panel, but as a tab inside AXIS:

```
$ cd configs/sim/axis/gladevc
$ linuxcnc gladevc_tab.ini
```



To run this panel inside *Touchy*:

```
$ cd configs/sim/touchy/gladevcp
$ linuxcnc gladevcp_touchy.ini
```



Functionally these setups are identical - they only differ in screen real estate requirements and visibility. Since it is possible to run several GladeVCP components in parallel (with different HAL component names), mixed setups are possible as well - for instance a panel on the right hand side, and one or more tabs for less-frequently used parts of the interface.

10.3.2.1 Exploring the example panel

While running `configs/sim/axis/gladevcp_panel.ini` or `configs/sim/axis/gladevcp_tab.ini`, explore *Show HAL Configuration* - you will find the `gladevcp` HAL component and may observe their pin values while interacting with the widgets in the panel. The HAL setup can be found in `configs/axis/gladevcp/manual-example.hal`.

The example panel has two frames at the bottom. The panel is configured so that resetting ESTOP activates the Settings frame and turning the machine on enables the Commands frame at the bottom. The HAL widgets in the Settings frame are linked to LEDs and labels in the Status frame, and to the current and prepared tool number - play with them to see the effect. Executing the `T<toolnumber>` and `M6` commands in the MDI window will change the current and prepared tool number fields.

The buttons in the *Commands* frame are *MDI Action widgets* - pressing them will execute an MDI command in the interpreter. The third button *Execute Oword subroutine* is an advanced example - it takes several HAL pin values from the Settings frame, and passes them as parameters to the Oword subroutine. The actual parameters received by the routine are displayed by `(DEBUG,)` commands - see `././nc_files/oword.ngc` for the subroutine body.

To see how the panel is integrated into AXIS, see the `[DISPLAY]GLADEVCP` statement in `configs/sim/axis/gladevcp/gladevcp_panel.ini`, the `[DISPLAY]EMBED*` statement in `configs/sim/axis/gladevcp/gladevcp_tab.ini` and `[HAL]POSTGUI HALFILE` statements in both `configs/sim/axis/gladevcp/gladevcp_tab.ini` and `configs/sim/axis/gladevcp/gladevcp_panel.ini`.

10.3.2.2 Exploring the User Interface description

The user interface is created with the glade UI editor - to explore it, you need to have [glade installed](#). To edit the user interface, run the command

```
$ glade configs/axis/gladevcp/manual-example.ui
```

The required glade program may be named glade-gtk2 on more recent systems.

The center window shows the appearance of the UI. All user interface objects and support objects are found in the right top window, where you can select a specific widget (or by clicking on it in the center window). The properties of the selected widget are displayed, and can be changed, in the right bottom window.

To see how MDI commands are passed from the MDI Action widgets, explore the widgets listed under *Actions* in the top right window, and in the right bottom window, under the *General* tab, the *MDI command* property.

10.3.2.3 Exploring the Python callback

See how a Python callback is integrated into the example:

- in glade, see the hits label widget (a plain GTK+ widget)
- in the button1 widget, look at the *Signals* tab, and find the signal *pressed* associated with the handler *on_button_press*
- in hitcounter.py, see the method *on_button_press* and see how it sets the label property in the *hits* object

This is just touching upon the concept - the callback mechanism will be handled in more detail in the [GladeVCP Programming](#) section.

10.3.3 Creating and Integrating a Glade user interface

10.3.3.1 Prerequisite: Glade installation

To view or modify Glade UI files, you need glade 3.38.2 or later installed - it is not needed just to run a GladeVCP panel. If the glade command is missing, install it with the command:

```
$ sudo apt install glade
```

Then verify installed version, which must be equal or superior to 3.6.7:

```
$ glade --version
```

Glade contains an internal Python interpreter, and only python3 is supported. This is true for Debian Bullseye, Ubuntu 21 and Mint 21 or later. Older versions will not work, you will get a python error.

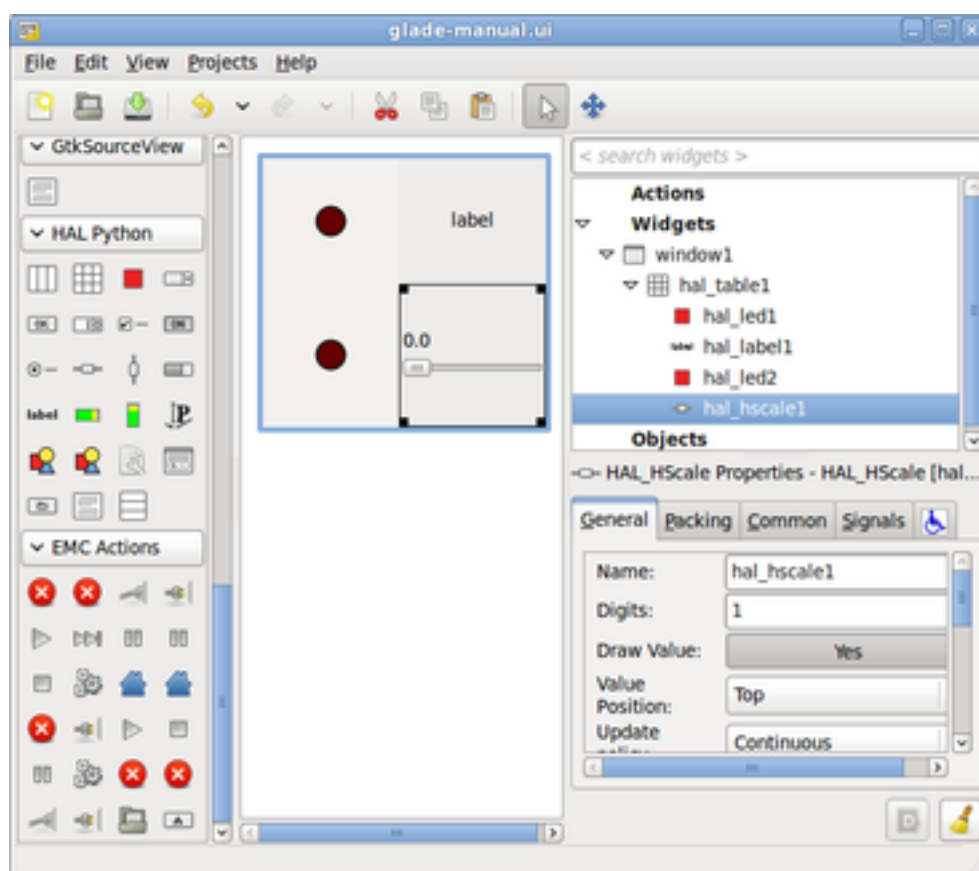
10.3.3.2 Running Glade to create a new user interface

This section just outlines the initial LinuxCNC-specific steps. For more information and a tutorial on glade, see <http://glade.gnome.org>. Some glade tips & tricks may also be found on [youtube](#).

Either modify an existing UI component by running `glade <file>.ui` or start a new one by just running the glade command from the shell.

- If LinuxCNC was not installed from a package, the LinuxCNC shell environment needs to be set up with `<linuxcncdir>/scripts/rip-environment`, otherwise glade won't find the LinuxCNC-specific widgets.
- When asked for unsaved preferences, just accept the defaults and hit *Close*.
- From *Toplevel* (left pane), pick *Window* (first icon) as top level window, which by default will be named *window1*. Do not change this name - GladeVCP relies on it.
- In the left tab, scroll down and expand *HAL Python* and *VCP Actions*.
- add a container like a *HAL_Box* or a *HAL_Table* from *HAL Python* to the frame
- pick and place some elements like LED, button, etc. within a container

This will look like so:



Glade tends to write a lot of messages to the shell window, which mostly can be ignored. Select *File*→*Save as*, give it a name like *myui.ui* and make sure it's saved as *GtkBuilder* file (radio button left bottom corner in Save dialog). GladeVCP will also process the older *libglade* format correctly but there is no point in using it. The convention for GtkBuilder file extension is *.ui*.

10.3.3.3 Testing a panel

You're now ready to give it a try (while LinuxCNC, e.g. AXIS is running) it with:

```
gladevcp myui.ui
```

GladeVCP creates a HAL component named like the basename of the UI file - *myui* in this case - unless overridden by the `-c <component name>` option. If running AXIS, just try *Show HAL configuration* and inspect its pins.

You might wonder why widgets contained a *HAL_Hbox* or *HAL_Table* appear greyed out (inactive). HAL containers have an associated HAL pin which is off by default, which causes all contained widgets to render inactive. A common use case would be to associate these container HAL pins with `halui.machine.is-on` or one of the `halui.mode.` signals, to assure some widgets appear active only in a certain state.

To just activate a container, execute the HAL command `setp gladevcp.<container-name> 1`.

10.3.3.4 Preparing the HAL command file

The suggested way of linking HAL pins in a GladeVCP panel is to collect them in a separate file with extension *.hal*. This file is passed via the `POSTGUI_HALFILE=` option in the HAL section of your INI file.



Caution

Do not add the GladeVCP HAL command file to the AXIS `[HAL]HALFILE=` ini section, this will not have the desired effect - see the following sections.

10.3.3.5 Integrating into AXIS, like PyVCP

Place the GladeVCP panel in the righthand side panel by specifying the following in the INI file:

```
[DISPLAY]
# add GladeVCP panel where PyVCP used to live:
GLADEVCP= -u ./hitcounter.py ./manual-example.ui

[HAL]
# HAL commands for GladeVCP components in a tab must be executed via POSTGUI_HALFILE
POSTGUI_HALFILE = ./manual-example.hal

[RS274NGC]
# gladevcp Demo specific 0word subs live here
SUBROUTINE_PATH = ../../nc_files/gladevcp_lib
```

The default HAL component name of a GladeVCP application started with the `GLADEVCP` option is: `gladevcp`.

The command line actually run by AXIS in the above configuration is as follows:

```
halcmd loadusr -Wn gladevcp gladevcp -c gladevcp -x {XID} -u ./hitcounter.py ./manual-
example.ui
```

You may add arbitrary `gladevcp` options here, as long as they don't collide with the above command line options.

It is possible to create a custom HAL component name by adding the `-c` option:

```
[DISPLAY]
# add GladeVCP panel where PyVCP used to live:
GLADEVCP= -c example -u ./hitcounter.py ./manual-example.ui
```

The command line actually run by AXIS for the above is:

```
halcmd loadusr -Wn example gladevcp -c example -x {XID} -u ./hitcounter.py ./manual-example
.ui
```

Note

The file specifiers like `./hitcounter.py`, `./manual-example.ui`, etc. indicate that the files are located in the same directory as the INI file. You might have to copy them to your directory (alternatively, specify a correct absolute or relative path to the file(s))

Note

The `[RS274NGC]SUBROUTINE_PATH=` option is only set so the example panel will find the `Oword` subroutine (`oword.ngc`) for the MDI Command widget. It might not be needed in your setup. The relative path specifier `../nc_files/gladevcp_lib` is constructed to work with directories copied by the configuration picker and when using a run-in-place setup.

10.3.3.6 Embedding as a Tab

To do so, edit your INI file and add to the `DISPLAY` and `HAL` sections of INI file as follows:

```
[DISPLAY]
# add GladeVCP panel as a tab next to Preview/DRO:
EMBED_TAB_NAME=GladeVCP demo
EMBED_TAB_COMMAND=halcmd loadusr -Wn gladevcp gladevcp -c gladevcp -x {XID} -u ./gladevcp/
hitcounter.py ./gladevcp/manual-example.ui

[HAL]
# HAL commands for GladeVCP components in a tab must be executed via POSTGUI_HALFILE
POSTGUI_HALFILE = ./gladevcp/manual-example.hal

[RS274NGC]
# gladevcp Demo specific Oword subs live here
SUBROUTINE_PATH = ../nc_files/gladevcp_lib
```

Note the *halcmd loadusr* way of starting the tab command - this assures that *POSTGUI_HALFILE* will only be run after the HAL component is ready. In rare cases you might run a command here which uses a tab but does not have an associated HAL component. Such a command can be started without *halcmd loadusr*, and this signifies to AXIS that it does not have to wait for a HAL component since there is none.

When changing the component name in the above example, note that the names used in `-Wn <component>` and `-c <component>` must be identical.

Try it out by running AXIS - there should be a new tab called *GladeVCP demo* near the DRO tab. Select that tab, you should see the example panel nicely fit within AXIS.

Note

Make sure the UI file is the last option passed to GladeVCP in both the `GLADEVCP=` and `EMBED_TAB_COMMAND=` statements.

10.3.3.7 Integrating into Touchy

To do add a GladeVCP tab to *Touchy*, edit your INI file as follows:

```
[DISPLAY]
# add GladeVCP panel as a tab
EMBED_TAB_NAME=GladeVCP demo
EMBED_TAB_COMMAND=gladevcp -c gladevcp -x {XID} -u ./hitcounter.py -H ./gladevcp-touchy.hal ←
./manual-example.ui

[RS274NGC]
# gladevcp Demo specific 0word subs live here
SUBROUTINE_PATH = ../../nc_files/gladevcp_lib
```

Note

The file specifiers like `./hitcounter.py`, `./manual-example.ui`, etc. indicate that the files are located in the same directory as the INI file. You might have to copy them to your directory (alternatively, specify a correct absolute or relative path to the file(s))

Note the following differences to the AXIS tab setup:

- The HAL command file is slightly modified since *Touchy* does not use the *halui* components so its signals are not available and some shortcuts have been taken.
- there is no `POSTGUI_HALFILE=` INI option, but passing the HAL command file on the `EMBED_TAB_COMMAND` line is ok
- the `halcmd loaduser -Wn ...` incantation is not needed.

10.3.4 GladeVCP command line options

See also *man gladevcp*. These are the GladeVCP command line options:

Usage: `gladevcp [options] myfile.ui`

Options:

- h, --help::
Show this help message and exit.
 - c NAME::
Set component name to NAME. Default is base name of UI file.
 - d::
Enable debug output
 - g GEOMETRY::
Set geometry WIDTHxHEIGHT+XOFFSET+YOFFSET. Values are in pixel units, XOFFSET/YOFFSET is referenced from top left of screen.
Use -g WIDTHxHEIGHT for just setting size or -g +XOFFSET+YOFFSET for just position
 - H FILE::
Execute HAL statements from FILE with `halcmd` after the component is set up and ready
 - m MAXIMUM::
Force panel window to maximize. Together with the -g geometry option one can move the panel to a second monitor and force it to use all of the screen
-

```
-t THEME::
    Set gtk theme. Default is system theme. Different panels can have different themes.
    An example theme can be found in the http://wiki.linuxcnc.org/cgi-bin/wiki.pl? ↩
    GTK_Themes[EMC Wiki].

-x XID::
    Re-parent GladeVCP into an existing window XID instead of creating a
    new top level window

-u FILE::
    Use File's as additional user defined modules with handlers

-U USEROPT::
    pass USEROPTs to Python modules
```

10.3.5 Understanding the GladeVCP startup process

The integration steps outlined above look a bit tricky, and they are. It does therefore help to understand the startup process of LinuxCNC and how this relates to GladeVCP.

The normal LinuxCNC startup process does the following:

- The realtime environment is started.
- All HAL components are loaded.
- The HAL components are linked together through the .hal cmd scripts.
- Task, iocontrol and eventually the user interface is started.
- Pre-GladeVCP the assumption was: by the time the UI starts, all of HAL is loaded, plumbed and ready to go.

The introduction of GladeVCP brought the following issue:

- GladeVCP panels need to be embedded in a master GUI window setup, e.g., AXIS, or Touchy, Gscreen, or Gmoccapy (embedded window or as an embedded tab).
- This requires the master GUI to run before the GladeVCP window can be hooked into the master GUI.
- However, GladeVCP is also a HAL component, and creates HAL pins of its own.
- As a consequence, all HAL plumbing involving GladeVCP HAL pins as source or destination must be run **after** the GUI has been set up.

This is the purpose of the POSTGUI_HALFILE. This INI option is inspected by the GUIs. If a GUI detects this option, it runs the corresponding HAL file after any embedded GladeVCP panel is set up. However, it does not check whether a GladeVCP panel is actually used, in which case the HAL cmd file is just run normally. So if you do NOT start GladeVCP through GLADEVCP or EMBED_TAB etc, but later in a separate shell window or some other mechanism, a HAL command file in POSTGUI_HALFILE will be executed too early. Assuming GladeVCP pins are referenced herein, this will fail with an error message indicating that the GladeVCP HAL component is not available.

So, in case you run GladeVCP from a separate shell window (i.e., not started by the GUI in an embedded fashion):

- You cannot rely on the POSTGUI_HALFILE INI option causing the HAL commands being run *at the right point in time*, so comment that out in the INI file.
- Explicitly pass the HAL command file which refers to GladeVCP pins to GladeVCP with the *-H <hal-cmd file>* option (see previous section).

10.3.6 HAL Widget reference

GladeVcp includes a collection of Gtk widgets with attached HAL pins called HAL Widgets, intended to control, display or otherwise interact with the LinuxCNC HAL layer. They are intended to be used with the Glade user interface editor. With proper installation, the HAL Widgets should show up in Glade's *HAL Python* widget group. Many HAL specific fields in the Glade *General* section have an associated mouse-over tool tip.

HAL signals come in two variants, bits and numbers. Bits are off/on signals. Numbers can be "float", "s32" or "u32". For more information on HAL data types see the [HAL manual](#). The GladeVcp widgets can either display the value of the signal with an indicator widget, or modify the signal value with a control widget. Thus there are four classes of GladeVcp widgets that you can connect to a HAL signal. Another class of helper widgets allow you to organize and label your panel.

- Widgets for indicating "bit" signals: [HAL_LED](#)
- Widgets for controlling "bit" signals: [HAL_Button](#) [HAL_RadioButton](#) [HAL_CheckButton](#)
- Widgets for indicating "number" signals: [HAL_Label](#), [HAL_ProgressBar](#), [HAL_HBar](#) and [HAL_VBar](#), [HAL_Meter](#)
- Widgets for controlling "number" signals: [HAL_SpinButton](#), [HAL_HScale](#) and [HAL_VScale](#), [Jog Wheel](#), [Speed Control](#)
- Sensitive control widgets: [State_Sensitive_Table](#) [HAL_Table](#) and [HAL_HBox](#)
- Tool Path preview: [HAL_Gremlin](#)
- Widgets to show axis positions: [DRO Widget](#), [Combi DRO Widget](#)
- Widgets for file handling: [IconView File Selection](#)
- Widgets for display/edit of all axes offsets: [OffsetPage](#)
- Widgets for display/edit of all tool offsets: [Tooloffset editor](#)
- Widget for G-code display and edit: [HAL_Sourceview](#)
- Widget for MDI input and history display: [MDI History](#)

10.3.6.1 Widget and HAL pin naming

Most HAL widgets have a single associated HAL pin with the same HAL name as the widget (glade: General→Name).

Exceptions to this rule currently are:

- [HAL_Spinbutton](#) and [HAL_ComboBox](#), which have two pins: a <widgetname>-f (float) and a <widgetname>-s32 pin
 - [HAL_ProgressBar](#), which has a <widgetname>-value input pin, and a <widgetname>-scale input pin.
-

10.3.6.2 Python attributes and methods of HAL Widgets

HAL widgets are instances of `GtkWidgets` and hence inherit the methods, properties and signals of the applicable `GtkWidget` class. For instance, to figure out which `GtkWidget`-related methods, properties and signals a `HAL_Button` has, lookup the description of [GtkButton](#) in the [PyGtk Reference Manual](#).

An easy way to find out the inheritance relationship of a given HAL widget is as follows: run glade, place the widget in a window, and select it; then choose the *Signals* tab in the *Properties* window. For example, selecting a `HAL_LED` widget, this will show that a `HAL_LED` is derived from a `GtkWidget`, which in turn is derived from a `GtkObject`, and eventually a `GObject`.

HAL Widgets also have a few HAL-specific Python attributes:

hal_pin

the underlying HAL pin Python object in case the widget has a single pin type

hal_pin_s, hal_pin_f

the s32 and float pins of the `HAL_Spinbutton` and `HAL_ComboBox` widgets - note these widgets do not have a `hal_pin` attribute!

hal_pin_scale

the float input pin of `HAL_ProgressBar` widget representing the maximum absolute value of input.

There are several HAL-specific methods of HAL Widgets, but the only relevant method is:

<halpin>.get()

Retrieve the value of the current HAL pin, where `<halpin>` is the applicable HAL pin name listed above.

10.3.6.3 Setting pin and widget values

As a general rule, if you need to set a HAL output widget's value from Python code, do so by calling the underlying `Gtk setter` (e.g. `set_active()`, `set_value()`) - do not try to set the associated pin's value by `halcomp[pinname] = value` directly because the widget will not take notice of the change!

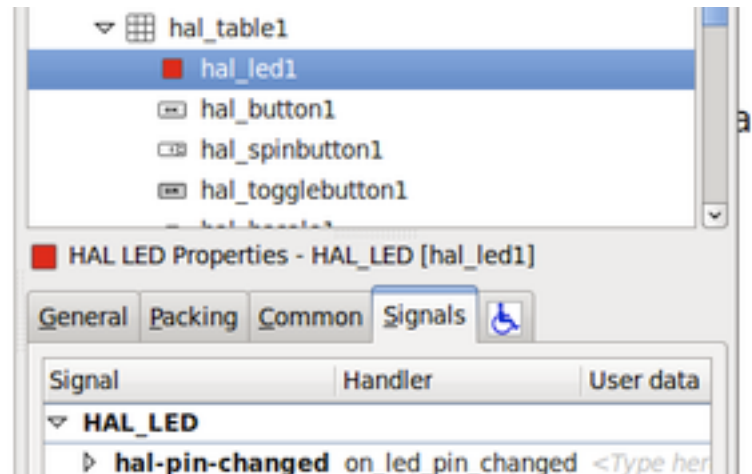
It might be tempting to *set HAL widget input pins* programmatically. Note this defeats the purpose of an input pin in the first place - it should be linked to, and react to signals generated by other HAL components. While there is currently no write protection on writing to input pins in HAL Python, this doesn't make sense. You might use `setp pinname` value in the associated HAL file for testing though.

It is perfectly OK to set an output HAL pin's value with `halcomp[pinname] = value` provided this HAL pin is not associated with a widget, that is, has been created by the `hal_glib.GPin(halcomp.newpin(<name>)` method (see [GladeVCP Programming](#) for an example).

10.3.6.4 The hal-pin-changed signal

Event-driven programming means that the UI tells your code when "something happens" - through a callback, like when a button was pressed. The output HAL widgets (those which display a HAL pin's value) like LED, Bar, VBar, Meter etc, support the *hal-pin-changed* signal which may cause a callback into your Python code when - well, a HAL pin changes its value. This means there's no more need for permanent polling of HAL pin changes in your code, the widgets do that in the background and let you know.

Here is an example how to set a `hal-pin-changed` signal for a `HAL_LED` in the Glade UI editor:



The example in `configs/apps/gladevcg/complex` shows how this is handled in Python.

10.3.6.5 Buttons

This group of widgets are derived from various Gtk buttons and consists of HAL_Button, HAL_ToggleButton, HAL_RadioButton and CheckButton widgets. All of them have a single output BIT pin named identical to the widget. Buttons have no additional properties compared to their base Gtk classes.

- HAL_Button: instantaneous action, does not retain state. Important signal: pressed
- HAL_ToggleButton, HAL_CheckButton: retains on/off state. Important signal: toggled
- HAL_RadioButton: a one-of-many group. Important signal: toggled (per button).
- Important common methods: `set_active()`, `get_active()`
- Important properties: `label`, `image`

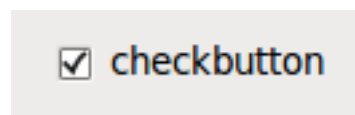


Figure 10.31: Check button

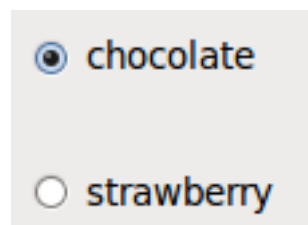


Figure 10.32: Radio buttons

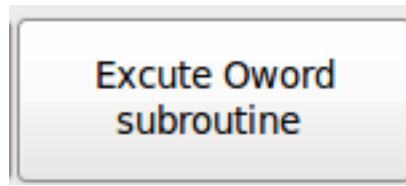


Figure 10.33: Toggle button

Tip

Defining radio button groups in Glade:

- Decide on default active button.
- In the other button's *General*→*Group* select the default active button's name in the *Choose a Radio Button in this project* dialog.

See `configs/apps/gladevcp/by-widget/` for a GladeVCP applications and UI file for working with radio buttons.

10.3.6.6 Scales

HAL_HScale and HAL_VScale are derived from the GtkHScale and GtkVScale respectively.

<widgetname>
out FLOAT pin

<widgetname>-s
out s32 pin

To make a scale useful in Glade, add an *Adjustment* (General → Adjustment → New or existing adjustment) and edit the adjustment object. It defines the default/min/max/increment values. Also, set adjustment *Page size* and *Page increment* to zero to avoid warnings.

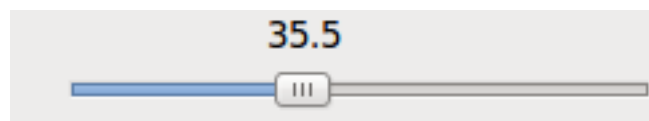


Figure 10.34: Example HAL_HScale

10.3.6.7 SpinButton

HAL SpinButton is derived from GtkSpinButton and holds two pins:

<widgetname>-f
out FLOAT pin

<widgetname>-s
out s32 pin

To be useful, Spinbuttons need an adjustment value like scales, see above.

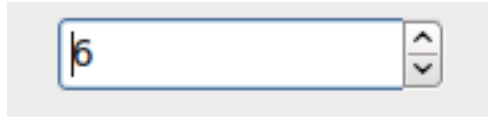


Figure 10.35: Example SpinButton

10.3.6.8 Hal_Dial

The `hal_dial` widget simulates a jogwheel or adjustment dial.

It can be operated with the mouse. You can just use the mouse wheel, while the mouse cursor is over the `Hal_Dial` widget, or you hold the left mouse button and move the cursor in circular direction to increase or decrease the counts.

By double clicking the left or right button the scale factor can be increased or decreased.

- Counterclockwise = reduce counts
- Clockwise = increase counts
- Wheel up = increase counts
- Wheel down = reduce counts
- left Double Click = x10 scale
- Right Double Click = /10 scale

`hal_dial` exports it's count value as HAL pins:

<widgetname>

out s32 pin

<widgetname>-scaled

out FLOAT pin

<widgetname>-delta-scaled

out FLOAT pin

`hal_dial` has the following properties:

cpr

Sets the Counts per Revolution, allowed values are in the range from 25 to 360
default = 100

show_counts

Set this to False, if you want to hide the counts display in the middle of the widget.
default = True

label

Set the content of the label which may be shown over the counts value.
If the label given is longer than 15 Characters, it will be cut to 15 Characters.
default = blank

center_color

This allows one to change the color of the wheel. It uses a GDK color string.
default = #bdefbdefbdef (gray)

count_type_shown

There are three counts available 0) Raw CPR counts 1) Scaled counts 2) Delta scaled counts.
default = 1

- count is based on the CPR selected - it will count positive and negative. It is available as a s32 pin.
- Scaled-count is CPR count times the scale - it can be positive and negative.
If you change the scale the output will immediately reflect the change. It is available as a FLOAT pin.
- Delta-scaled-count is cpr count CHANGE, times scale.
If you change the scale, only the counts after that change will be scaled and then added to the current value.
It is available as a FLOAT pin.

scale_adjustable

Set this to False if you want to disallow scale changes by double clicking the widget.
If this is false the scale factor will not show on the widget.
default = True

scale

Set this to scale the counts.
default = 1.0

There are ways to directly control the widget using Python.

Using goobject to set the above listed properties:

```
[widget name].set_property("cpr",int(value))
[widget name].set_property("show_counts", True)
[widget name].set_property("center_color",gtk.gdk.Color('#bdefbdefbdef'))
[widget name].set_property('label', 'Test Dial 12345')
[widget name].set_property('scale_adjustable', True)
[widget name].set_property('scale', 10.5)
[widget name].set_property('count_type_shown', 0)
```

There are python methods:

- [widget name].get_value()
Will return the counts value as a s32 integer
- [widget name].get_scaled_value()
Will return the counts value as a float
- [widget name].get_delta_scaled_value()
Will return the counts value as a float
- [widget name].set_label("string")
Sets the label content with "string"

There are two GObject signals emitted:

- count_changed
Emitted when the widget's count changes eg. from being wheel scrolled.
- scale_changed
Emitted when the widget's scale changes eg. from double clicking.

Connect to these like so:


```
[widget name].connect('count_changed', [count function name])
[widget name].connect('scale_changed', [scale function name])
```

The callback functions would use this pattern:

```
def [count function name](widget, count,scale,delta_scale):
```

This will return: the widget, the current count, scale and delta scale of that widget.

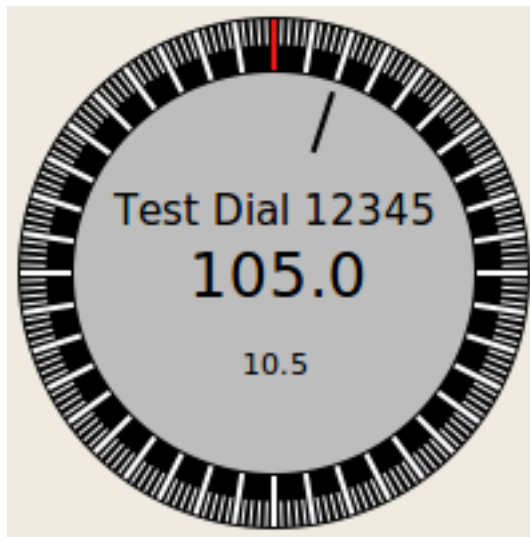


Figure 10.36: Example Hal_Dial

10.3.6.9 Jog Wheel

The jogwheel widget simulates a real jogwheel. It can be operated with the mouse. You can just use the mouse wheel, while the mouse cursor is over the JogWheel widget, or you push the left mouse button and move the cursor in circular direction to increase or decrease the counts.

- Counterclockwise = reduce counts
- Clockwise = increase counts
- Wheel up = increase counts
- Wheel down = reduce counts

As moving the mouse the drag and drop way may be faster than the widget can update itself, you may lose counts turning too fast. It is recommended to use the mouse wheel, and only for very rough movements the drag and drop way.

jogwheel exports its count value as HAL pin:

```
<widgetname>-s
  out s32 pin
```

jogwheel has the following properties:

size

Sets the size in pixel of the widget, allowed values are in the range of 100 to 500 default = 200

cpr

Sets the Counts per Revolution, allowed values are in the range from 25 to 100 default = 40

show_counts

Set this to False, if you want to hide the counts display in the middle of the widget.

label

Set the content of the label which may be shown over the counts value. The purpose is to give the user an idea about the usage of that jogwheel. If the label given is longer than 12 Characters, it will be cut to 12 Characters.

There a couple ways to directly control the widget using Python.

Using gobject to set the above listed properties:

```
[widget name].set_property("size",int(value))  
[widget name].set_property("cpr",int(value))  
[widget name].set_property("show_counts, True)
```

There are two python methods:

- [widget name].get_value()
Will return the counts value as integer
- [widget name].set_label("string")
Sets the label content with "string"

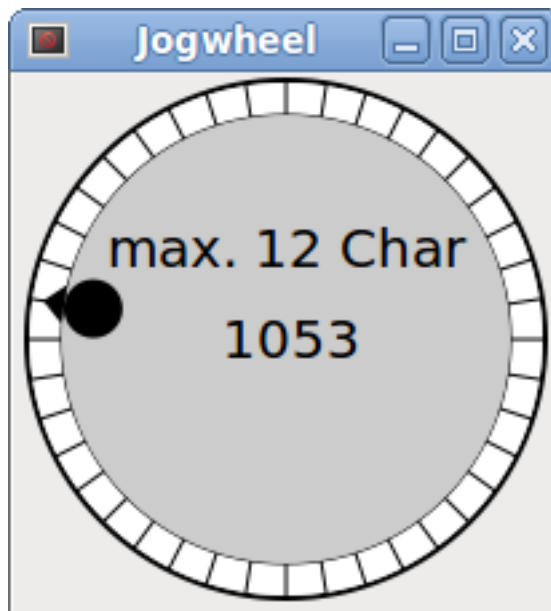


Figure 10.37: Example JogWheel

10.3.6.10 Speed Control

`speedcontrol` is a widget specially made to control an adjustment with a touch screen. It is a replacement to the normal scale widget which is difficult to slide on a touch screen.

The value is controlled with two button to increase or decrease the value. The increment will change as long a button is pressed. The value of each increment as well as the time between two changes can be set using the widget properties.

speedcontrol offers some HAL pin:

<widgetname>-value

out float pin

The shown value of the widget.

<widgetname>-scaled-value

out float pin

The shown value divided by the scale value, this is very useful, if the velocity is shown in units / min, but LinuxCNC expects it to be in units / second.

<widgetname>-scale

in float pin

The scale to apply.

Default is 60.

<widgetname>-increase

in bit pin

As long as the pin is true, the value will increase.

Very handy with connected momentary switch.

<widgetname>-decrease

in bit pin

As long as the pin is true, the value will decrease.

Very handy with connected momentary switch.

speedcontrol has the following properties:

height

Integer

The height of the widget in pixel.

Allowed values are 24 to 96.

Default is 36.

value

Float

The start value to set.

Allowed values are in the range from 0.001 to 99999.0.

Default is 10.0.

min

Float

The min allowed value.

Allowed values are 0.0 to 99999.0.

Default is 0.0.

If you change this value, the increment will be reset to default, so it might be necessary to set afterwards a new increment.

max

Float

The max allowed value.

Allowed values are 0.001 to 99999.0.

Default is 100.0.

If you change this value, the increment will be reset to default, so it might be necessary to set afterwards a new increment.

increment

Float

Sets the applied increment per mouse click.

Allowed values are 0.001 to 99999.0 and -1.

Default is -1, resulting in 100 increments from min to max.

inc_speed

Integer

Sets the timer delay for the increment speed holding pressed the buttons.

Allowed values are 20 to 300.

Default is 100.

unit

String

Sets the unit to be shown in the bar after the value.

Any string is allowed.

Default is "".

color

Color

Sets the color of the bar.

Any hex color is allowed.

Default is "#FF8116".

template

String

Text template to display the value. Python formatting is used.

Any allowed format.

Default is "%.1f".

do_hide_button

Boolean

Whether to show or hide the increment and decrement button.

True or False.

Default = False.

There are a couple ways to directly control the widget using Python.

Using GObject to set the above listed properties:

```
[widget name].set_property("do_hide_button",bool(value))
[widget name].set_property("color","#FF00FF")
[widget name].set_property("unit", "mm/min")
etc.
```

There are also Python methods to modify the widget:

```
[widget name].set_adjustment(gtk-adjustment)
```

You can assign an existing adjustment to the control, that way it is easy to replace existing sliders without many code changes. Be aware, that after changing the adjustment you may need to set a new increment, as it will be reset to its default (100 steps from MIN to MAX):

- `[widget name].get_value()`
Will return the counts value as float
- `[widget name].set_value(float(value))`
Sets the widget to the commanded value
- `[widget name].set_digits(int(value))`
Sets the digits of the value to be used

- `[widget name].hide_button(bool(value))`
Hide or show the button

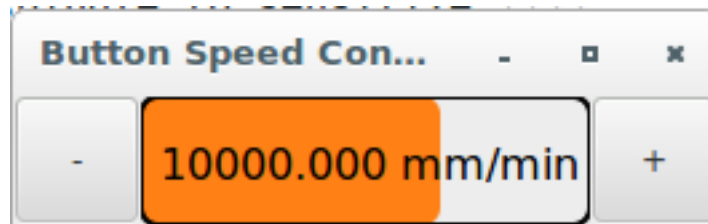


Figure 10.38: Example Speedcontrol

10.3.6.11 Label

`hal_label` is a simple widget based on `GtkLabel` which represents a HAL pin value in a user-defined format.

label_pin_type

The pin's HAL type (0:s32, 1:float, 2:u32), see also the tooltip on 'General→HAL pin type '(note this is different from PyVCP which has three label widgets, one for each type).

text_template

Determines the text displayed - a Python format string to convert the pin value to text. Defaults to `%s` (values are converted by the `str()` function) but may contain any legit as an argument to Python's `format()` method.

Example: Distance: `%.03f` will display the text and the pin value with 3 fractional digits padded with zeros for a FLOAT pin.

10.3.6.12 Containers

- `HAL_HideTable`
- `HAL_Table State_Sensitive_Table`
- `HAL_HBox`

These containers are meant to be used to insensitize (grey out) or hide their children. Insensitized children will not respond to input.

`hal_hidetable` has one HAL BIT input pin which controls if it's child widgets are hidden or not.

<Panel_basename>.<widgetname>

FIXME pin name If the pin is low then child widgets are visible which is the default state.

`hal_table` and `hal_hbox` have one HAL BIT input pin which controls if their child widgets are sensitive or not.

These widgets's pin name uses the convention:

<Panel_basename>.<widgetname>

FIXME pin name If the pin is low then child widgets are inactive which is the default state.

`state_sensitive_table` responds to the state to LinuxCNC's interpreter.
 Optionally selectable to respond to *must-be-all-homed*, *must-be-on* and *must-be-idle*.
 You can combine them. It will always be insensitive at Estop.



Warning

HAL_Hbox is deprecated - use HAL_Table.

If current panels use it, it won't fail. You just won't find it in the GLADE editor anymore.
 Future versions of GladeVCP may remove this widget completely and then you will need to update the panel.

Tip

If you find some part of your GladeVCP application is *grayed out* (insensitive), see whether a HAL_Table pin is unset or unconnected.

10.3.6.13 LED

The `hal_led` simulates a real indicator LED.
 It has a single input BIT pin which controls it's state: ON or OFF.
 LEDs have several properties which control their look and feel:

on_color

String defining ON color of LED.
 May be any valid gdk.Color name.
 Not working on Ubuntu 8.04.

off_color

String defining OFF color of LED.
 May be any valid gdk.Color name or special value dark. dark means that OFF color will be set to 0.4 value of ON color.
 Not working on Ubuntu 8.04.

pick_color_on, pick_color_off

Colors for ON and OFF states.
 These may be represented as #RRRRGGGGBBBB strings and are optional properties which have precedence over `on_color` and `off_color`.

led_size

LED radius (for square - half of LED's side)

led_shape

LED Shape.
 Valid values are 0 for round, 1 for oval and 2 for square shapes.

led_blink_rate

If set and LED is ON then it's blinking.
 Blink period is equal to "led_blink_rate" specified in milliseconds.

create_hal_pin

Select/deselect creation of a HAL pin to control the LED.
 With no HAL pin created LED can be controlled with a python function.

As an input widget, LED also supports the `hal-pin-changed` signal. If you want to get a notification in your code when the LED's HAL pin was changed, then connect this signal to a handler, for example `on_led_pin_changed` and provide the handler as follows:

```
def on_led_pin_changed(self,hal_led,data=None):
    print("on_led_pin_changed() - HAL pin value:",hal_led.hal_pin.get())
```

This will be called at any edge of the signal and also during program start up to report the current value.



Figure 10.39: Example LEDs

10.3.6.14 ProgressBar

Note

This widget might go away.
Use the HAL_HBar and HAL_VBar widgets instead.

The HAL_ProgressBar is derived from gtk.ProgressBar and has two float HAL input pins:

<widgetname>

the current value to be displayed

<widgetname>-scale

the maximum absolute value of input

HAL_ProgressBar has the following properties:

scale

Value scale.

Sets the maximum absolute value of input. Same as setting the <widgetname>.scale pin.

A float, range from -2^{24} to $+2^{24}$.

green_limit

Green zone lower limit

yellow_limit

Yellow zone lower limit

red_limit

Red zone lower limit

text_template

Text template to display the current value of the <widgetname> pin.

Python formatting may be used for dict {"value":value}



Figure 10.40: Example HAL_ProgressBar

10.3.6.15 ComboBox

HAL_ComboBox is derived from gtk.ComboBox. It enables choice of a value from a dropdown list.

HAL_ComboBox exports two HAL pins:

<widgetname>-f

Current value, type FLOAT

<widgetname>-s

Current value, type s32

HAL_ComboBox has the following property which can be set in Glade:

column

The column index.

Type s32.

Valid range from -1..100.

Defaults value -1.

In default mode this widget sets the pins to the index of the chosen list entry. So if your widget has three labels, it may only assume values 0,1 and 2.

In column mode (column > -1), the value reported is chosen from the ListStore array as defined in Glade. So typically your widget definition would have two columns in the ListStore, one with text displayed in the dropdown, and an int or float value to use for that choice.

There's an example in configs/apps/by-widget/combobox.{py,ui} which uses column mode to pick a float value from the ListStore.

If you're confused like me about how to edit ComboBox ListStores and CellRenderer, see http://www.youtube.com/watch?v=Z5_FrW2cL8.

10.3.6.16 Bars

HAL_Bar and HAL_VBar widgets for horizontal and vertical bars representing float values.

HAL_Bar and HAL_VBar each have one input FLOAT HAL pin.

HAL_Bar and HAL_VBar both bars have the following properties:

invert

Swap min and max direction.

An inverted HBar grows from right to left, an inverted VBar from top to bottom.

min, max

Minimum and maximum value of desired range. It is not an error condition if the current value is outside this range.

show limits

Used to select/deselect the limits text on bar.

zero

Zero point of range.

If it's inside of min/max range then the bar will grow from that value and not from the left (or right) side of the widget.

Useful to represent values that may be both positive or negative.

force_width, force_height

Forced width or height of widget.

If not set then size will be deduced from packing or from fixed widget size and bar will fill whole area.

text_template

Like in Label, sets text format for min/max/current values.

Can be used to turn off value display.

value

Sets the bar display to the value entered.

Used only for testing in GLADE editor.

The value will be set from a HAL pin.

target_value

Sets the target line to the value entered.

Used only for testing in GLADE editor.

The value will can be set in a Python function.

target_width

Width of the line that marks the target value.

bg_color

Background (inactive) color of bar.

target_color

Color of the the target line.

z0_color, z1_color, z2_color

Colors of different value zones.

Defaults are green, yellow and red.

For description of zones see `z*_border` properties.

z0_border, z1_border

Define up bounds of color zones.

By default only one zone is enabled. If you want more then one zone set `z0_border` and `z1_border` to desired values so zone 0 will fill from 0 to first border, zone 1 will fill from first to second border and zone 2 from last border to 1.

Borders are set as fractions.

Valid values range from 0 to 1.

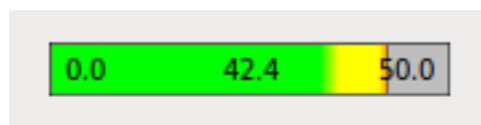


Figure 10.41: Horizontal bar

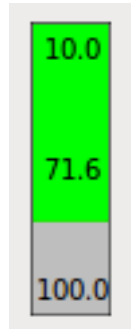


Figure 10.42: Vertical bar

10.3.6.17 Meter

HAL_Meter is a widget similar to PyVCP meter - it represents a float value.

HAL_Meter has one input FLOAT HAL pin.

HAL Meter has the following properties:

min, max

Minimum and maximum value of desired range.

It is not an error condition if the current value is outside this range.

force_size

Forced diameter of widget.

If not set then size will be deduced from packing or from fixed widget size, and meter will fill all available space with respect to aspect ratio.

text_template

Like in Label, sets text format for current value.

Can be used to turn off value display.

label

Large label above center of meter.

sublabel

Small label below center of meter.

bg_color

Background color of meter.

z0_color, z1_color, z2_color

Colors of different value zones.

Defaults are green, yellow and red.

For description of zones see z*_border properties.

z0_border, z1_border

Define up bounds of color zones.

By default only one zone is enabled. If you want more then one zone set z0_border and z1_border to desired values so zone 0 will fill from min to first border, zone 1 will fill from first to second border and zone 2 from last border to max.

Borders are set as values in range min-max.

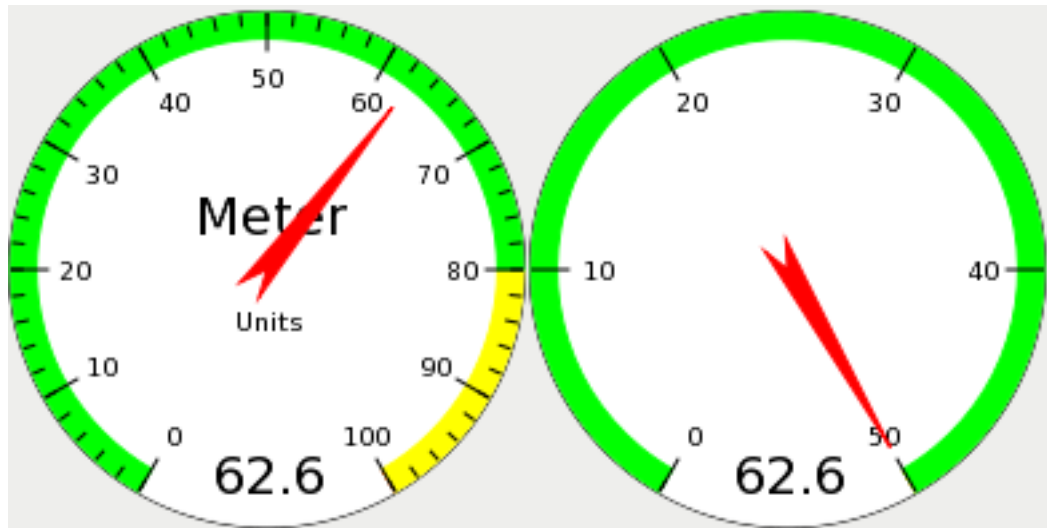


Figure 10.43: Example HAL Meters

10.3.6.18 HAL_Graph

This widget is for plotting values over time.

10.3.6.19 Gremlin tool path preview for .ngc files

Gremlin is a plot preview widget similar to the AXIS preview window. It assumes a running LinuxCNC environment like AXIS or Touchy. To connect to it, inspects the `INI_FILE_NAME` environment variable. Gremlin displays the current .ngc file - it does monitor for changes and reloads the ngc file if the file name in AXIS/Touchy changes. If you run it in a GladeVCP application when LinuxCNC is not running, you might get a traceback because the Gremlin widget can't find LinuxCNC status, like the current file name.

Gremlin does not export any HAL pins.

Gremlin has the following properties:

show tool speed

This displays the tool speed.
Default = true.

show commanded

This selects the DRO to use: commanded or actual values.
Default = true.

use metric units

This selects the DRO to use: metric or imperial units.
Default = true.

show rapids

This tells the plotter to show the rapid moves.
Default = true.

show DTG

This selects the DRO to display the distance-to-go value.
Default = true.

show relative

This selects the DRO to show values relative to user system or machine coordinates.
Default = true.

show live plot

This tells the plotter to draw or not.
Default = true.

show limits

This tells the plotter to show the machine's limits.
Default = true.

show lathe radius

This selects the DRO to display the X axis in radius or diameter, if in lathe mode (selectable in the INI file with LATHE = 1).
Default = true.

show extents

This tells the plotter to show the extents.
Default = true.

show tool

This tells the plotter to draw the tool.
Default = true.

show program

TODO

use joints mode

Used in non trivialkins machines (e.g., robots).
Default = false.

grid size

Sets the size of the grid (only visible in the X, Y and Z views).
Defaults to 0

use default mouse controls

This disables the default mouse controls.
This is most useful when using a touchscreen as the default controls do not work well. You can programmatically add controls using python and the handler file technique.
Default = true.

view

May be any of x, y, y2, z, z2, p (perspective).
Defaults to z view.

enable_dro

Type = boolean.
Whether to draw a DRO on the plot or not.
Default = true.

mouse_btn_mode

Type = integer.
Mouse button handling: leads to different functions of the button:

- 0 = default: left rotate, middle move, right zoom
 - 1 = left zoom, middle move, right rotate
 - 2 = left move, middle rotate, right zoom
 - 3 = left zoom, middle rotate, right move
 - 4 = left move, middle zoom, right rotate
-

- 5 = left rotate, middle zoom, right move
- 6 = left move, middle zoom, right zoom

Mode 6 is recommended for plasmas and lathes, as rotation is not needed for such machines.

There are a couple ways to directly control the widget using Python.

Using goobject to set the above listed properties:

```
[widget name].set_property('view','P')
[widget name].set_property('metric_units',False)
[widget name].set_property('use_default_controls',False)
[widget name].set_property('enable_dro',False)
[widget name].set_property('show_program',False)
[widget name].set_property('show_limits',False)
[widget name].set_property('show_extents_option',False)
[widget name].set_property('show_live_plot',False)
[widget name].set_property('show_tool',False)
[widget name].set_property('show_lathe_radius',True)
[widget name].set_property('show_dtg',True)
[widget name].set_property('show_velocity',False)
[widget name].set_property('mouse_btn_mode',4)
```

There are python methods:

```
[widget name].show_offsets = True
[widget name].grid_size = .75
[widget name].select_fire(event.x,event.y)
[widget name].select_prime(event.x,event.y)
[widget name].start_continuous_zoom(event.y)
[widget name].set_mouse_start(0,0)
[widget name].gremlin.zoom_in()
[widget name].gremlin.zoom_out()
[widget name].get_zoom_distance()
[widget name].set_zoom_distance(dist)
[widget name].clear_live_plotter()
[widget name].rotate_view(x,y)
[widget name].pan(x,y)
```

Hints

- If you set all the plotting options false but show_offsets true you get an offsets page instead of a graphics plot.
- If you get the zoom distance before changing the view then reset the zoom distance, it's much more user friendly.
- if you select an element in the preview, the selected element will be used as rotation center point

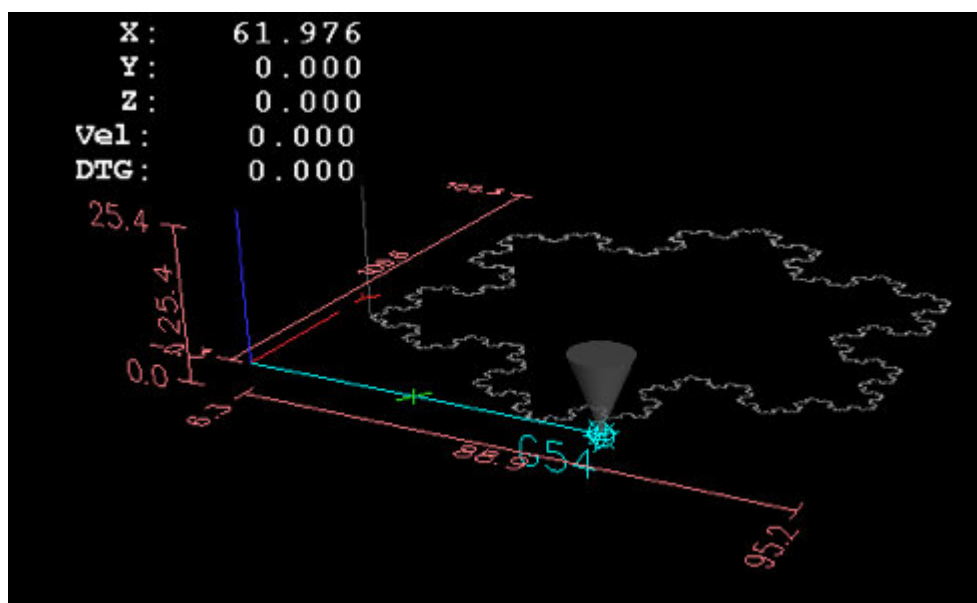


Figure 10.44: Gremlin Example

10.3.6.20 HAL_Offset

The HAL_Offset widget is used to display the offset of a single axis.

HAL Offset has the following properties:

Joint Number

Used to select which axis (technically which joint) is displayed.

On a trivialkins machine (mill, lathe, router) axis vrs joint number are:

0:X 1:Y 2:Z 3:A 4:B 5:C 6:U 7:V 8:W

Text template for metric units

You can use python formatting to display the position with different precision.

Text template for imperial units

You can use python formatting to display the position with different precision.

Reference Type

0:G5x 1:tool 2:G92 3:Rotation around Z

10.3.6.21 DRO widget

The DRO widget is used to display the current axis position.

It has the following properties:

Actual Position

select actual (feedback) position or commanded position.

Text template for metric units

You can use python formatting to display the position with different precision.

Text template for imperial units

You can use python formatting to display the position with different precision.

Reference Type

- absolute ([machine origin](#)), or
- relative (to current user coordinate origin - G5x), or
- distance-to-go (relative to current user coordinate origin).

Joint Number

Used to select which axis (technically which joint) is displayed.

On a trivialkins machine (mill, lathe, router) axis vrs joint number are:

0:X 1:Y 2:Z 3:A 4:B 5:C 6:U 7:V 8:W

Display units

Used to toggle the display units between metric and imperial.

Hints

- If you want the display to be right justified, set the X align to 1.0
- If you want different colors or size or text change the attributes in the glade editor (eg scale is a good way to change the size of the text)
- The background of the widget is actually see through - so if you place if over an image the DRO numbers will show on top of it with no background. There is a special technique to do this. See the animated function diagrams below.
- The DRO widget is a modified gtk label widget. As such much or what can be done to a gtk label can be done to DRO widget.

There are a couple ways to directly control the widget using Python.

Using goobject to set the above listed properties:

```
[widget name].set_property("display_units_mm",True)
[widget name].set_property("actual",True)
[widget name].set_property("mm_text_template","%f")
[widget name].set_property("imperial_text_template","%f")
[widget name].set_property("Joint_number",3)
[widget name].set_property("reference_type",3)
```

There are two python methods:

```
[widget name].set_dro_inch()
[widget name].set_dro_metric()
```

10.3.6.22 Combi_DRO widget

The Combi_DRO widget is used to display the current, the relative axis position and the distance to go in one DRO.

By clicking on the DRO the Order of the DRO will toggle around.

In Relative Mode the actual coordinate system will be displayed.

Combi_DRO has the following properties:

joint_number

Used to select which axis (technically which joint) is displayed.

On a trivialkins machine (mill, lathe, router) axis vrs. joint number are:

0:X 1:Y 2:Z etc

actual

Select actual (feedback) or commanded position.

metric_units

Used to toggle the display units between metric and imperial.

auto_units

Units will toggle between metric and imperial according to the active G-code being G20 or G21.

Default is TRUE.

diameter

Whether to display position as diameter or radius.

In diameter mode the DRO will display the joint value multiplied by 2.

mm_text_template

You can use python formatting to display the position with different precision.

Default is "%10.3f".

imperial_text_template

You can use python formatting to display the position with different precision.

Default is "%9.4f".

homed_color

The foreground color of the DRO numbers if the joint is homed.

Default is green.

unhomed_color

The foreground color of the DRO numbers if the joint is not homed.

Default is red.

abs_color

The background color of the DRO, if main DRO shows absolute coordinates.

Default is blue.

rel_color

The background color of the DRO, if main DRO shows relative coordinates.

Default is black.

dtg_color

The background color of the DRO, if main DRO shows distance to go.

Default is yellow.

font_size

The font size of the big numbers, the small ones will be 2.5 times smaller.

The value must be an integer in the range of 8 to 96.

Default is 25.

toggle_readout

A left mouse click will toggle the DRO readout through the different modes ["Rel", "Abs", "DTG"].

By unchecking the box you can disable that behavior. The toggling can still be done with [widget name].toggle_readout().

Value must be boolean.

Default is TRUE.

cycle_time

The time the DRO waits between two polls.

this setting should only be changed if you use more than 5 DRO at the same time, i.e., on a 6 axis config, to avoid, that the DRO slows down the main application too much.

The value must be an integer in the range of 100 to 1000. FIXME unit=ms ?

Default is 150.

Using gobject to set the above listed properties:

```
[widget name].set_property(property, value)
```

There are several python methods to control the widget:

- `[widget name].set_to_inch(state)`
Sets the DRO to show imperial units.
state = boolean (True or False)
Default is FIXME.
- `[widget name].set_auto_units(state)`
If True the DRO will change units according to active G-code (G20 / G21).
state = boolean (True or False)
Default is True.
- `[widget name].set_to_diameter(state)`
If True the DRO will show the diameter not the radius, i.e., the axis value multiplied by 2 (specially needed for lathes).
state = boolean (True or False)
Default is False.
- `[widget name].toggle_readout()`
Toggles the order of the DRO in the widget.
- `[widget name].change_axisletter(letter)`
Changes the automatically given axis letter.
Very useful to change an lathe DRO from *X* to *R* or *D*.
letter = string
- `[widget name].get_order()`
Returns the order of the DRO in the widget mainly used to maintain them consistent.
The order will also be transmitted with the clicked signal.
Returns a list containing the order.
- `[widget name].set_order(order)`
Sets the order of the DRO, mainly used to maintain them consistent.
order = list object, must be one of:
 - ["Rel", "Abs", "DTG"] (default)
 - ["DTG", "Rel", "Abs"]
 - ["Abs", "DTG", "Rel"]
- `[widget name].get_position()`
Returns the position of the DRO as a list of floats.
The order is independent of the order shown on the DRO and will be given as [Absolute , relative , DTG].
 - Absolute = the machine coordinates, depends on the actual property will give actual or commanded position.

- Relative = will be the coordinates of the actual coordinate system.
- DTG = the distance to go.
Will mostly be 0, as this function should not be used while the machine is moving, because of time delays.

The widget will emit the following signals:

- `clicked`
This signal is emitted, when the user has clicked on the Combi_DRO widget.
It will send the following data:
 - `widget` = widget object
The widget object that sends the signal.
 - `joint_number` = integer
The joint number of the DRO, where *0:X 1:Y 2:Z etc.*
 - `order` = list object
The order of the DRO in that widget.
The order may be used to set other Combi_DRO widgets to the same order with `[widget name].set_order`
- `units_changed`
This signal is emitted if the DRO units are changed.
It will send the following data:
 - `widget` = widget object
The widget object that sends the signal.
 - `metric_units` = boolean
True if the DRO does display metric units, False in case of imperial display.
- `system_changed`
This signal is emitted if the DRO units are changed.
It will send the following data:
 - `widget` = widget object
The widget object that sends the signal.
 - `system` = string
The actual coordinate system. Will be one of G54 G55 G56 G57 G58 G59 G59.1 G59.2 G59.3 or Rel if non has been selected at all, what will only happen in Glade with no LinuxCNC running.

There are some information you can get through commands, which may be of interest for you:

- `[widget name].system`
The actual system, as mentioned in the `system_changed` signal.
 - `[widget name].homed`
True if the joint is homed.
 - `[widget name].machine_units`
0 if Imperial, 1 if Metric.
-



Figure 10.45: Example: Three Combi_DRO in a window

X = Relative Mode
 Y = Absolute Mode
 Z = DTG Mode

10.3.6.23 IconView (File Select)

This is a touch screen friendly widget to select a file and to change directories.

IconView widget has the following properties:

icon_size

Sets the size of the displayed icon.
 Allowed values are integers in the range from 12 to 96.
 Default is 48.

start_dir

Sets the directory to start in when the widget is shown first time.
 Must be a string, containing a valid directory path.
 Default is "/".

jump_to_dir

Sets the "jump to" directory, which is selected by the corresponding button in the bottom button list (the 5th button counting from the left).
 Must be a string, containing a valid directory path.
 Default is "~".

filetypes

Sets the file filter for the objects to be shown.
 Must be a string containing a comma separated list of extensions to be shown.
 Default is "ngc,py".

sortorder

Sets the sorting order of the displayed icon.
 Must be an integer value from 0 to 3, where:

- 0 = ASCENDING (sorted according to file names)
- 1 = DESCENDING (sorted according to file names)
- 2 = FOLDERFIRST (show the folders first, then the files), default
- 3 = FILEFIRST (show the files first, then the folders)

Using goobject to set the above listed properties:

```
[widget name].set_property(property,Value)
```

There are python methods to control the widget:

- `[widget name].show_buttonbox(state)`
If False the bottom button box will be hidden.
This is helpful in custom screens, with special buttons layouts to not alter the layout of the GUI.
Good example for that is gmocapy.
state = boolean (True or False).
Default is True.
- `[widget name].show_filelabel(state)`
If True the file label (between the IconView window and the bottom button box) will be shown.
Hiding this label may save place, but showing it is very useful for debugging reasons.
state = boolean (True or False).
Default is True.
- `[widget name].set_icon_size(iconsize)`
Sets the icon size.
Must be an integer in the range from 12 to 96.
Default = 48.
- `[widget name].set_directory(directory)`
Allows to set an directory to be shown.
directory = string (a valid file path).
- `[widget name].set_filetypes(filetypes)`
Sets the file filter to be used.
Only files with the given extensions will be shown.
filetypes = string containing a comma separated list of extensions.
Default = "ngc,py".
- `[widget name].get_selected()`
Returns the path of the selected file, or None if a directory has been selected.
- `[widget name].refresh_filelist()`
Refreshes the filelist.
Needed if you add a file without changing the directory.

If the button box has been hidden, you can reach the functions of this button through it's clicked signals like so:

```
[widget name].btn_home.emit("clicked")
[widget name].btn_jump_to.emit("clicked")
[widget name].btn_sel_prev.emit("clicked")
[widget name].btn_sel_next.emit("clicked")
[widget name].btn_get_selected.emit("clicked")
[widget name].btn_dir_up.emit("clicked")
[widget name].btn_exit.emit("clicked")
```

The widget will emit the following signals:

- **selected**

This signal is emitted when the user selects an icon.

It will return a string containing a file path if a file has been selected, or None if a directory has been selected.

- **sensitive**

This signal is emitted when the buttons change their state from sensitive to not sensitive or vice versa.

This signal is useful to maintain surrounding GUI synchronized with the button of the widget. See gmoccap as example.

It will return the **buttonname** and the new **state**:

- buttonname is one of btn_home, btn_dir_up, btn_sel_prev, btn_sel_next, btn_jump_to or btn_select

- state is a boolean and will be True or False.

- **exit**

This signal is emitted when the exit button has been pressed to close the IconView.

Mostly needed if the application is started as stand alone.

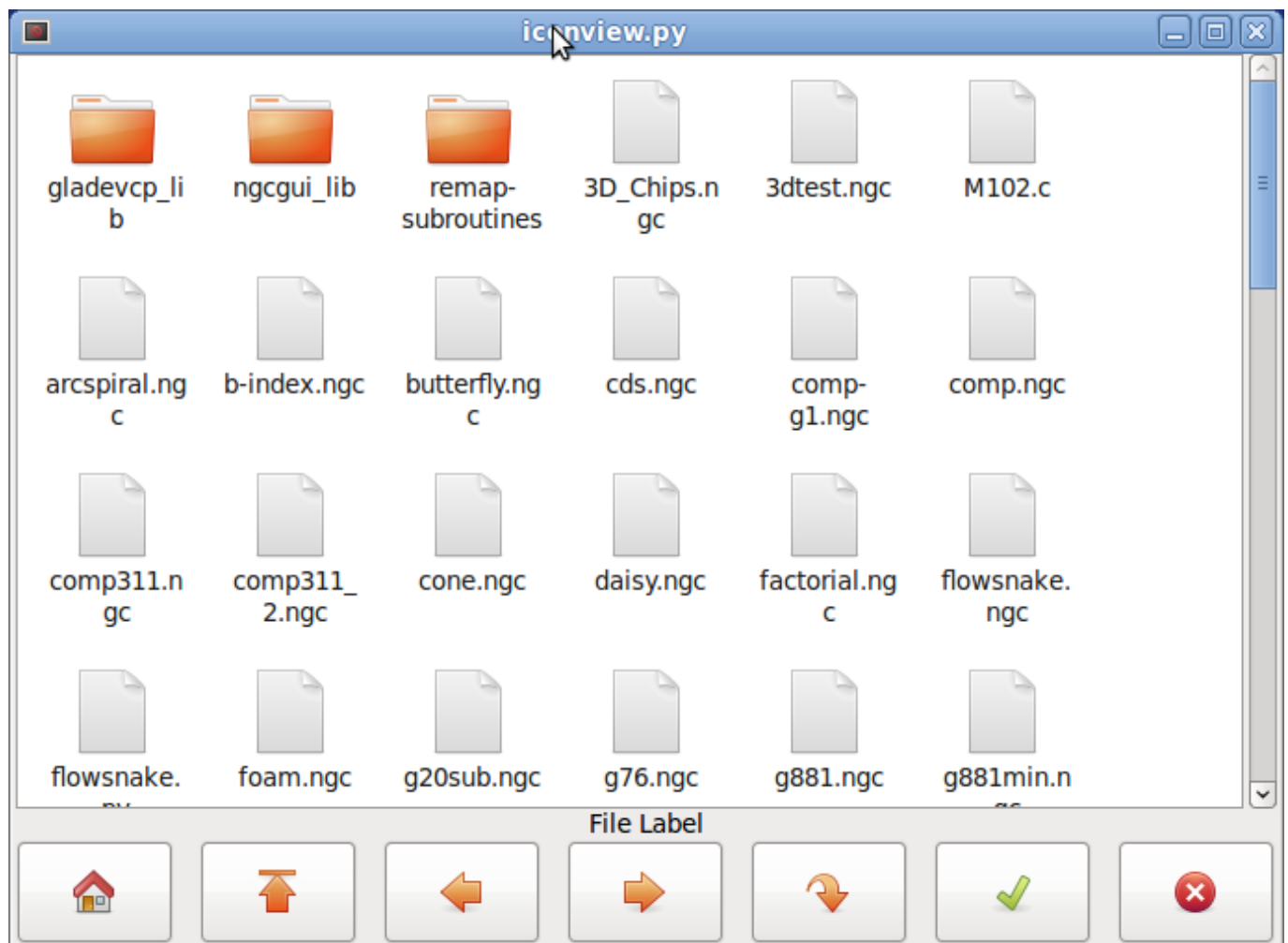


Figure 10.46: Iconview Example

10.3.6.24 Calculator widget

This is a simple calculator widget, that can be used for numerical input. You can preset the display and retrieve the result or that preset value.

calculator has the following properties:

Is editable

This allows the entry display to be typed into from a keyboard.

Set Font

This allows you to set the font of the display.

There a couple ways to directly control the widget using Python.

Using goobject to set the above listed properties:

```
[widget name].set_property("is_editable",True)
[widget name].set_property("font","sans 25")
```

There are python methods:

- [widget name].set_value(2.5)
This presets the display and is recorded.
- [widget name].set_font("sans 25")
- [widget name].set_editable(True)
- [widget name].get_value()
Returns the calculated value - a float.
- [widget name].set_editable(True)

- [widget name].get_preset_value()
Returns the recorded value: a float.

10.3.6.25 Tooleditor widget

This is a tooleditor widget for displaying and modifying a tool file. If in lathe mode, it will display wear offsets and tool offsets separately. Wear offsets are designated by tool number above 10000 (Fanuc style). It checks the current file once a second to see if LinuxCNC updated it.

Note

LinuxCNC requires remapping of tool calls to actually use wear offsets.

tooleditor has the following properties:

Hidden Columns

This will hide the given columns.

The columns are designated (in order) as such: s,t,p,x,y,z,a,b,c,u,v,w,d,i,j,q.

You can hide any number of columns including the select and comments.

There are a couple ways to directly control the widget using Python.

Using goobject to set the above listed properties:

```
[widget name].set_properties('hide_columns','uvwijq')
```

This would hide the uvw and q columns and show all others.

There are python methods:

- `[widget name].set_visible("ijq",False)`
Would hide ij and Q columns and leave the rest as they were.
- `[widget name].set_filename(path_to_file)`
Sets and loads the tool file.
- `[widget name].reload(None)`
Reloads the current toolfile.
- `[widget name].set_font('sans 16,tab='1')`
Sets the (Pango) font on the Tab, column title, and tool data.
The `all_offsets`, `wear_offsets`, `tool_offsets` can be set at the same time by adding 1, 2 and/or 3 to the tab string.
Default is all the tabs set.
- `[widget name].set_title_font('sans 16,tab='1')`
Sets the (Pango) font on the column titles only.
The `all_offsets`, `wear_offsets`, `tool_offsets` can be set at the same time by adding 1, 2 and/or 3 to the tab string.
Default is all the tabs set.
- `[widget name].set_tab_font('sans 16,tab='1')`
Sets the (Pango) font on the tabs only.
The `all_offsets`, `wear_offsets`, `tool_offsets` can be set at the same time by adding 1, 2 and/or 3 to the tab string.
Default is all the tabs set.
- `[widget name].set_col_visible("abcUVW", False, tab='1')`
This would hide (False) the abcuvw columns on tab 1 (`all_offsets`)
- `[widget name].set_lathe_display(value)`
Hides or shows the wear and tool offset tabs used for lathes
- `[widget name].get_toolinfo(toolnum)`
Returns the tool information array of the requested toolnumber or current tool if no tool number is specified.
Returns None if tool not found in table or if there is no current tool.
- `[widget name].hide_buttonbox(self, True)`
Convenience method to hide buttons.
You must call this after `show_all()`.
- `[widget name].get_selected_tool()`
Return the user selected (highlighted) tool number.
- `[widget name].set_selected_tool(toolnumber)`
Selects (highlights) the requested tool.

Select	Tool#	Pocket	X	Y	Z	Diameter	Comments
<input type="checkbox"/>	2	0	1.4230	-1.5670	0.0000	0.0000	comment
<input type="checkbox"/>	1	4	1.2345	0.0000	0.4440	0.0000	comment
<input type="checkbox"/>	0	0	-5.1234	0.0000	0.0000	0.0000	comment
<input type="checkbox"/>	0	0	123.0000	0.0000	0.0000	0.0000	tool 1
<input checked="" type="checkbox"/>	0	0	45.6700	0.0000	1.0000	0.0000	drill

Delete Add Reload Apply
Cancel OK

Figure 10.47: Tooleditor Example

10.3.6.26 Offsetpage

The **Offsetpage** widget is used to display/edit the offsets of all the axes.

It has convenience buttons for zeroing G92 and Rotation-Around-Z offsets.

It will only allow you to select the edit mode when the machine is on and idle.

You can directly edit the offsets in the table at this time. Unselect the edit button to allow the **OffsetPage** to reflect changes.

It has the following properties:

Hidden Columns

A no-space list of columns to hide. The columns are designated (in order) as such: xyzabcuvwt. You can hide any of the columns.

Hidden Rows

A no-space list of rows to hide.

The rows are designated (in order) as such: 0123456789abc.

You can hide any of the rows.

Pango Font

Sets text font type and size.

HighLight color

When editing this is the highlight color.

Active color

When **OffsetPage** detects an active user coordinate system it will use this color for the text.

Text template for metric units

You can use python formatting to display the position with different precision.

Text template for imperial units

You can use python formatting to display the position with different precision.

There a couple ways to directly control the widget using Python.

Using goobject to set the above listed properties:


```
[widget name].set_property("highlight_color",gdk.Color('blue'))
[widget name].set_property("foreground_color",gdk.Color('black'))
[widget name].set_property("hide_columns","xyzabcuvwt")
[widget name].set_property("hide_rows","123456789abc")
[widget name].set_property("font","sans 25")
```

There are python methods to control the widget:

- `[widget name].set_filename("../../../configs/sim/gscreen/gscreen_custom/sim.var")`
- `[widget name].set_col_visible("Yabuvw",False)`
- `[widget name].set_row_visible("456789abc",False)`
- `[widget name].set_to_mm()`
- `[widget name].set_to_inch()`
- `[widget name].hide_button_box(True)`
- `[widget name].set_font("sans 20")`
- `[widget name].set_highlight_color("violet")`
- `[widget name].set_foreground_color("yellow")`
- `[widget name].mark_active("G55")`
Allows you to directly set a row to highlight, e.g., in case you wish to use your own navigation controls. See the chapter on [Gmoccapy](#).
- `[widget name].selection_mask = ("Tool","Rot","G5x")`
These rows are NOT selectable in edit mode.
- `[widget name].set_names([['G54', 'Default'], ["G55", "Vice1"], ['Rot', 'Rotational']])`
This allows you to set the text of the *T* column of each/any row.
This is a list of a list of offset-name/user-name pairs.
The default text is the same as the offset name.
- `[widget name].get_names()`
This returns a list of a list of row-keyword/user-name pairs.
The user name column is editable, so saving this list is user friendly.
See `set_names` above.

Offset	X	Y	Z	A	B	C	U	V	W	Offset Name
Tool	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	Tool
G5x	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	G5x
Rot			0.00							Rotation of Z
G92	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	G92
G54	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	G54
G55	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	G55
G56	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	G56
G57	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	G57
G58	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	G58
G59	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	G59
G59.1	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	G59.1
G59.2	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	G59.2
G59.3	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	G59.3

Zero G92

Zero Rotational

Edit

Cancel

OK

Figure 10.48: Offsetpage Example

10.3.6.27 HAL_sourceview widget

This is for displaying and simple editing of G-code. It looks for .ngc highlighting specs in ~/share/gtksourcece
The current running line will be highlighted.

With external python glue code it can:

- Search for text, undo and redo changes.
- Be used for program line selection.

There are python methods to control the widget:

- `[widget name].redo()`
Redo one level of changes.
- `[widget name].undo()`
Undo one level of changes
- `[widget name].text_search(direction=True,mixed_case=True,text='G92')`
Searches forward (`direction = True`) or backward,
Searches with mixed case (`mixed_case = True`) or exact match
- `[widget name].set_line_number(linenummer)`
Sets the line to highlight.
Uses the sourceview line numbers.
- `[widget name].get_line_number()`
Returns the currently highlighted line.
- `[widget name].line_up()`
Moves the highlighted line up one line.

- [widget name].line_down()
Moves the highlighted line down one line.
- [widget name].load_file('filename')
Loads a file.
Using None (not a filename string) will reload the same program.
- [widget name].get_filename()
FIXME description



Figure 10.49: Sourceview Example

10.3.6.28 MDI history

This is for displaying and entering MDI codes.

It will be automatically grayed out when MDI is not available, eg during E-stop and program running.

font_size_tree

Integer value between 8 and 96.

Will modify the default font size of the treeview to the selected value.

font_size_entry

Integer value between 8 and 96.

Will modify the default font size of the entry to the selected value.

use_double_click

Boolean, True enables the mouse double click feature and a double click on an entry will submit that command.

It is not recommended to use this feature on real machines, as a double click on a wrong entry may cause dangerous situations

Using goobject to set the above listed properties:

```
[widget name].set_property("font_size_tree",10)
[widget name].set_property("font_size_entry",20)
[widget name].set_property("use_double_click",False)
```

10.3.6.29 Animated function diagrams: HAL widgets in a bitmap

For some applications it might be desirable to have a background image - like a functional diagram - and position widgets at appropriate places in that image. A good combination is setting a bitmap background image, like from a .png file, making the GladeVCP window fixed-size, and use the glade Fixed widget to position widgets on this image.

The code for the below example can be found in `configs/apps/gladevcp/animated-backdrop`:

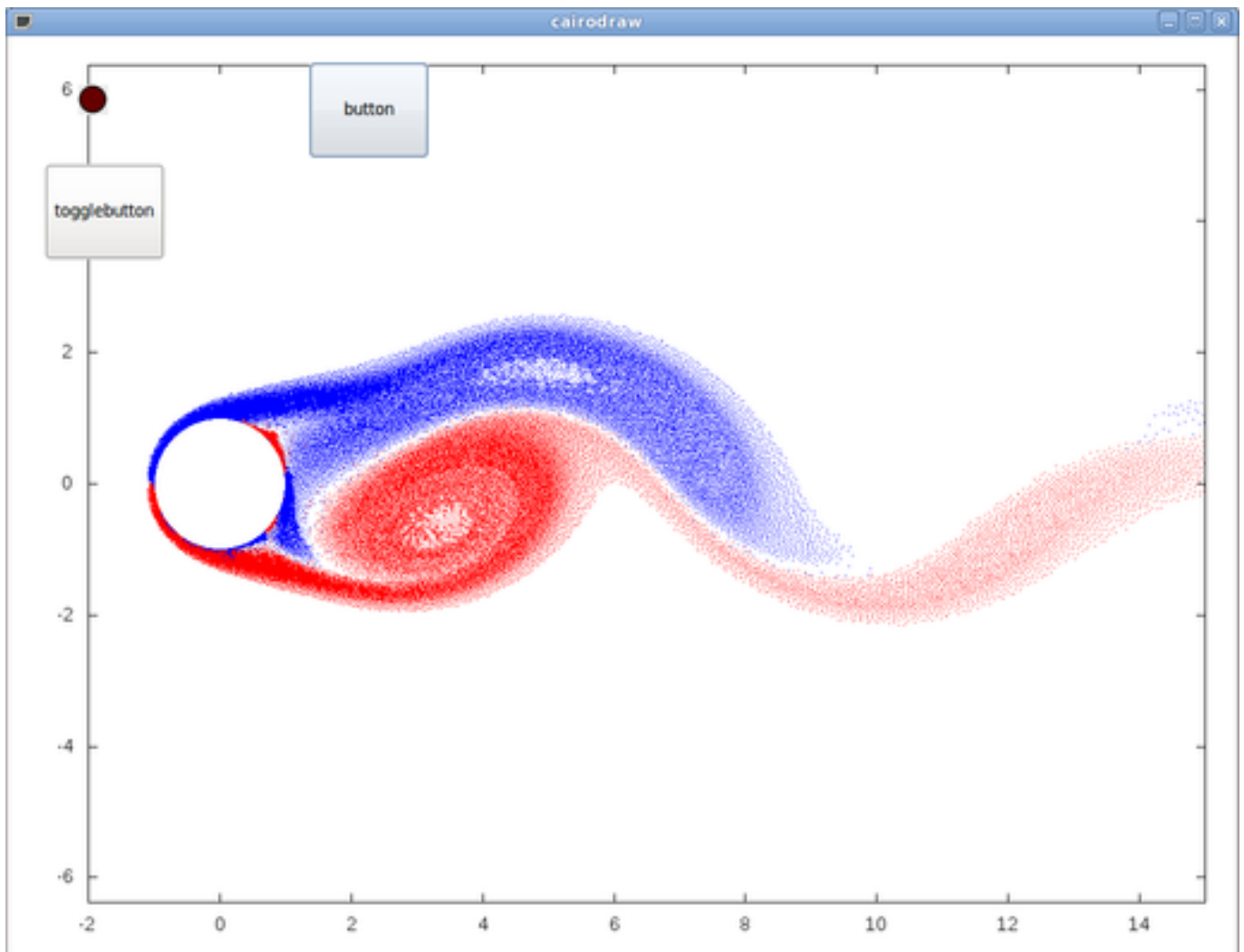


Figure 10.50: HAL widgets in a bitmap Example

10.3.7 Action Widgets Reference

GladeVcp includes a collection of "canned actions" called **VCP Action Widgets** for the Glade user interface editor.

Note

Other than HAL widgets, which interact with HAL pins, VCP Actions interact with LinuxCNC and the G-code interpreter.

VCP Action Widgets are derived from the `Gtk.Action` widget.

The Action widget in a nutshell:

- is an object available in Glade
- has no visual appearance by itself
- it's purpose: associate a visible, sensitive UI component like menu, toolbutton, button with a command. See these widget's *General*→*Related*→*Action* property.
- the "canned action" will be executed when the associated UI component is triggered (button press, menu click..)
- provides an easy way to execute commands without resorting to Python programming.

The appearance of VCP Actions in Glade is roughly as follows:



Figure 10.51: Action Widgets

Tooltip hovers provide a description.

10.3.7.1 VCP Action widgets

VCP Action widgets are one-shot type widgets. They implement a single action and are for use in simple buttons, menu entries or radio/check groups.

10.3.7.2 VCP Action python

This widget is used to execute small arbitrary python code. The command string may use special keywords to access important functions.

- GSTAT for access to the Gstat library that is used for LinuxCNC status
- STAT for access to LinuxCNC's status via the linuxcnc python module
- CMD for access to LinuxCNC's commands via the linuxcnc python module
- EXT for access to the handler file functions if available
- linuxcnc for access to the linuxcnc python module

- `self` for access to the widget instance

There are options to

- select when the widget will be active,
- set the mode before the command is executed.

Example command to just print a message to the terminal:

```
print('action activated')
```

Example command to set the machine to off state:

```
CMD.state(linuxcnc.STATE_OFF)
```

Example command to call a handler function that passes data:

```
EXT.on_button_press(self, 100)
```

You can use a colon to separate multiple commands.

```
print('Set Machine Off');CMD.state(linuxcnc.STATE_OFF)
```

10.3.7.3 VCP ToggleAction widgets

These are **bi-modal** widgets. They implement two actions or use a second (usually pressed) state to indicate that currently an action is running. Toggle actions are aimed for use in `ToggleButtons`, `ToggleToolButtons` or toggling menu items. A simplex example is the ESTOP toggle button.

Currently the following widgets are available:

- The ESTOP toggle sends ESTOP or ESTOP_RESET commands to LinuxCNC depending on it's state.
- The ON/OFF toggle sends STATE_ON and STATE_OFF commands.
- Pause/Resume sends AUTO_PAUSE or AUTO_RESUME commands.

The following toggle actions have only one associated command and use the pressed state to indicate that the requested operation is running:

- The Run toggle sends an AUTO_RUN command and waits in the pressed state until the interpreter is idle again.
- The Stop toggle is inactive until the interpreter enters the active state (is running G-code) and then allows user to send AUTO_ABORT command.
- The MDI toggle sends given MDI command and waits for its completion in pressed inactive state.

10.3.7.4 The Action_MDI Toggle and Action_MDI widgets

These widgets provide a means to execute arbitrary MDI commands.

The `Action_MDI` widget does not wait for command completion as the `Action_MDI Toggle` does, which remains disabled until command complete.

10.3.7.5 A simple example: Execute MDI command on button press

`configs/apps/gladevcp/mdi-command-example/whoareyou.ui` is a Glade UI file which conveys the basics:

1. Open it in Glade and study how it's done.
2. Start AXIS, and then start this from a terminal window with `gladevcp whoareyou.ui`.
3. See the `hal_action_mdil` action and it's MDI command property - this just executes (`MSG, "Hi, I'm an VCP_Action_MDI"`) so there should be a message popup in AXIS like so:

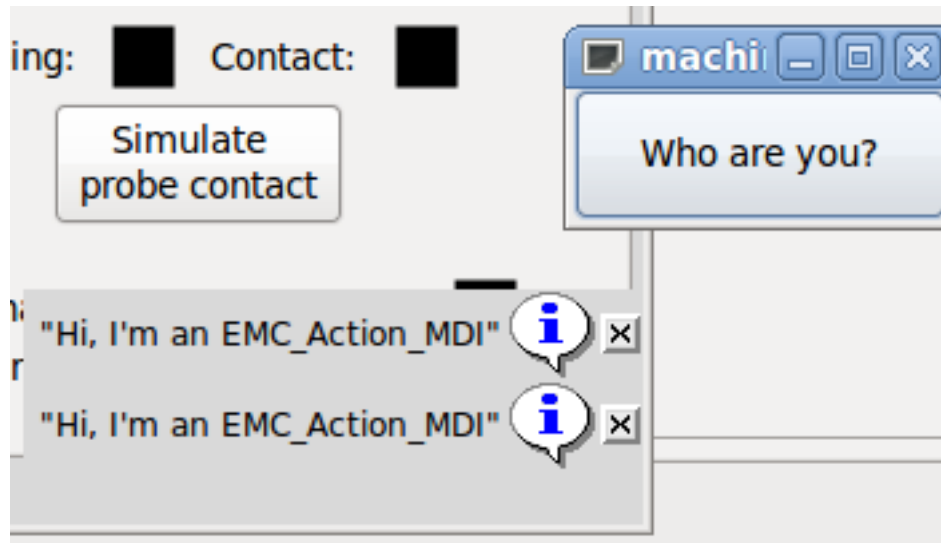


Figure 10.52: Action_MDI Simple Example

You'll notice that the button associated with the Action_MDI action is grayed out if the machine is off, in E-Stop or if the interpreter is running. It will automatically become active when the machine is turned on and out of E-Stop, and the program is idle.

10.3.7.6 Parameter passing with Action_MDI and ToggleAction_MDI widgets

Optionally, MDI command strings may have parameters substituted before they are passed to the interpreter. Parameters currently may be names of HAL pins in the GladeVCP component. This is how it works:

- assume you have a *HAL SpinBox* named `speed`, and you want to pass it's current value as a parameter in an MDI command.
- The HAL SpinBox will have a float-type HAL pin named `speed-f` (see *HalWidgets* description).
- To substitute this value in the MDI command, insert the HAL pin name enclosed like so: `${pin-name}`
- for the above HAL SpinBox, we could use (`MSG, "The speed is: ${speed-f}"`) just to show what's happening.

The example UI file is `configs/apps/gladevcp/mdi-command-example/speed.ui`. Here's what you get when running it:

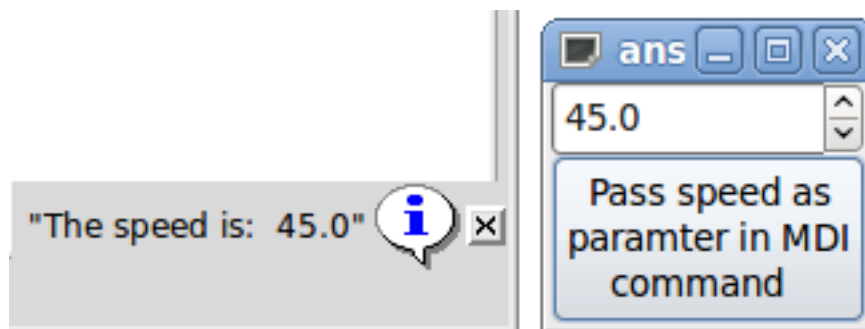


Figure 10.53: Action_MDI Parameter Passing Example

10.3.7.7 An advanced example: Feeding parameters to an O-word subroutine

It's perfectly OK to call an O-word subroutine in an MDI command, and pass HAL pin values as actual parameters. An example UI file is in `configs/apps/gladevc/mdi-command-example/owordsub.ui`.

Place `nc_files/gladevc_lib/oword.ngc` so AXIS can find it, and run `gladevc owordsub.ui` from a terminal window. This looks like so:

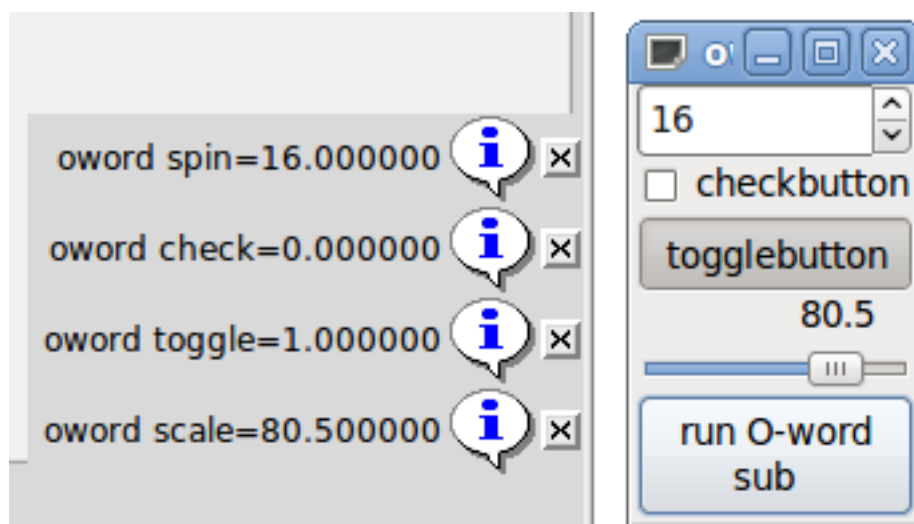


Figure 10.54: Action_MDI Advanced Example

10.3.7.8 Preparing for an MDI Action, and cleaning up afterwards

The LinuxCNC G-code interpreter has a single global set of variables, like feed, spindle speed, relative/absolute mode and others. If you use G-code commands or O-word subs, some of these variables might get changed by the command or subroutine - for example, a probing subroutine will very likely set the feed value quite low. With no further precautions, your previous feed setting will be overwritten by the probing subroutine's value.

To deal with this surprising and undesirable side effect of a given O-word subroutine or G-code statement executed with an LinuxCNC ToggleAction_MDI, you might associate pre-MDI and post-MDI handlers with a given LinuxCNC ToggleAction_MDI. These handlers are optional and provide a way to save any state before executing the MDI Action, and to restore it to previous values afterwards. The signal names are `mdi-command-start` and `mdi-command-stop`; the handler names can be set in Glade like any other handler.

Here's an example how a feed value might be saved and restored by such handlers (note that LinuxCNC command and status channels are available as `self.linuxcnc` and `self.stat` through the `VCP_ActionBase` class):

```
def on_mdi_command_start(self, action, userdata=None):
    action.stat.poll()
    self.start_feed = action.stat.settings[1]

def on_mdi_command_stop(self, action, userdata=None):
    action.linuxcnc.mdi('F%.1f' % (self.start_feed))
    while action.linuxcnc.wait_complete() == -1:
        pass
```

Only the `Action_MDI Toggle` widget supports these signals.

Note

In a later release of LinuxCNC, the new M-codes M70-M72 will be available.

They will make saving state before a subroutine call, and restoring state on return much easier.

10.3.7.9 Using the LinuxCNC Stat object to deal with status changes

Many actions depend on LinuxCNC status - is it in manual, MDI or auto mode? is a program running, paused or idle? You cannot start an MDI command while a G-code program is running, so this needs to be taken care of. Many LinuxCNC actions take care of this themselves, and related buttons and menu entries are deactivated when the operation is currently impossible.

When using Python event handlers - which are at a lower level than Actions - one needs to take care of dealing with status dependencies oneself. For this purpose, there's the LinuxCNC Stat widget: to associate LinuxCNC status changes with event handlers.

LinuxCNC Stat has no visible component - you just add it to your UI with Glade. Once added, you can associate handlers with its following signals:

- state-related:
 - state-estop: emitted when E-Stop condition occurs,
 - state-estop-reset: emitted when machine is reset,
 - state-on: emitted when machine is turned on,
 - state-off: emitted when machine is turned off.
 - mode-related:
 - mode-manual: emitted when LinuxCNC enters manual mode,
 - mode-mdi: emitted when LinuxCNC enters MDI mode,
 - mode-auto: emitted when LinuxCNC enters automatic mode,
 - interpreter-related: emitted when the G-code interpreter changes into that mode
 - interp-run
 - interp-idle
 - interp-paused
 - interp-reading
 - interp-waiting
 - file-loaded
-

- line-changed
- homing-related: emitted when LinuxCNC is homed or not
 - all-homed
 - not-all-homed

10.3.8 GladeVCP Programming

10.3.8.1 User Defined Actions

Most widget sets, and their associated user interface editors, support the concept of callbacks - functions in user-written code which are executed when *something happens* in the UI - events like mouse clicks, characters typed, mouse movement, timer events, window hiding and exposure and so forth.

HAL output widgets typically map input-type events like a button press to a value change of the associated HAL pin by means of such a - predefined - callback. Within PyVCP, this is really the only type of event handling supported - doing something more complex, like executing MDI commands to call a G-code subroutine, is not supported.

Within GladeVCP, HAL pin changes are just one type of the general class of events (called signals) in GTK+. Most widgets may originate such signals, and the Glade editor supports associating such a signal with a Python method or function name.

If you decide to use user-defined actions, your job is to write a Python module whose class methods - or in the simple case, just functions - can be referred to in Glade as event handlers. GladeVCP provides a way to import your module(s) at startup and will automatically link your event handlers with the widget signals as set in the Glade UI description.

10.3.8.2 Core Library

There are three libraries of functions that can be used to help program GladeVCP.

- *Info*: collects details from the INI file.
- *Action*: A collection of functions to change LinuxCNC states.
- *Status*: Reports the state of LinuxCNC. It wraps around *Gstat*.

Importing and instantiating the libraries:

```
from gladevcp.core import Info, Action
```

```
ACTION = Action()  
INFO = Info()
```

using the library functions:

```
print(INFO.MACHINE_IS_METRIC)  
ACTION.SET_ERROR_MESSAGE('Something went wrong')
```

10.3.8.3 An example: adding custom user callbacks in Python

This is just a minimal example to convey the idea - details are laid out in the rest of this section.

GladeVCP can not only manipulate or display HAL pins, you can also write regular event handlers in Python. This could be used, among others, to execute MDI commands. Here's how you do it:

Write a Python module like so and save as e.g. `handlers.py`:

```
nhits = 0
def on_button_press(gtkobj, data=None):
    global nhits
    nhits += 1
    gtkobj.set_label("hits: %d" % nhits)
```

In Glade, define a button or HAL button, select the *Signals* tab, and in the GtkButton properties select the *pressed* line. Enter `on_button_press` there, and save the Glade file.

Then add the option `-u handlers.py` to the GladeVCP command line. If your event handlers are spread over several files, just add multiple `-u <pyfilename>` options.

Now, pressing the button should change its label since it's set in the callback function.

What the `-u` flag does: all Python functions in this file are collected and setup as potential callback handlers for your Gtk widgets - they can be referenced from Glade *Signals* tabs. The callback handlers are called with the particular object instance as parameter, like the GtkButton instance above, so you can apply any GtkButton method from there.

Or do some more useful stuff, like calling an MDI command!

10.3.8.4 HAL value change events

HAL input widgets, like a LED, automatically associate their HAL pin state (on/off) with the optical appearance of the widget (LED lit/dark).

Beyond this built-in functionality, one may associate a change callback with any HAL pin, including those of predefined HAL widgets. This fits nicely with the event-driven structure of a typical widget application: every activity, be it mouse click, key, timer expired, or the change of a HAL pin's value, generates a callback and is handled by the same orthogonal mechanism.

For user-defined HAL pins not associated with a particular HAL widget, the signal name is *value-changed*. See the [Adding HAL pins](#) section below for details.

HAL widgets come with a pre-defined signal called *hal-pin-changed*. See the [HAL Widgets](#) section for details.

10.3.8.5 Programming model

The overall approach is as follows:

- Design your UI with Glade, and set signal handlers where you want actions associated with a widget.
- Write a Python module which contains callable objects (see *handler models* below).
- Pass your module's path name to GladeVCP with the `-u <module>` option.
- GladeVCP imports the module, inspects it for signal handlers and connects them to the widget tree.
- The main event loop is run.

For simple tasks it's sufficient to define functions named after the Glade signal handlers. These will be called when the corresponding event happens in the widget tree. Here's a trivial example - it assumes that the *pressed* signal of a Gtk Button or HAL Button is linked to a callback called *on_button_press*:

```
nhits = 0
def on_button_press(gtkobj,data=None):
    global nhits
    nhits += 1
    gtkobj.set_label("hits: %d" % nhits)
```

Add this function to a Python file and run as follows:

```
gladevcp -u <myhandler>.py mygui.ui
```

Note communication between handlers has to go through global variables, which does not scale well and is positively unpythonic. This is why we came up with the class-based handler model.

The idea here is: handlers are linked to class methods. The underlying class(es) are instantiated and inspected during GladeVCP startup and linked to the widget tree as signal handlers. So the task now is to write:

- one or more several class definition(s) with one or several methods, in one module or split over several modules,
- a function *get_handlers* in each module which will return a list of class instances to GladeVCP - their method names will be linked to signal handlers

Here is a minimum user-defined handler example module:

```
class MyCallbacks :
    def on_this_signal(self,obj,data=None):
        print("this_signal happened, obj=",obj)

def get_handlers(halcomp,builder,useropts):
    return [MyCallbacks ()]
```

Now, *on_this_signal* will be available as signal handler to your widget tree.

For GladeVCP panel which respond to HAL inputs it may be important that the handler code can tell that the GladeVCP panel is currently active and displayed. For example a panel inside the Touchy interface might well need to perform an action when the switch connected to touchy.cycle-start is operated (in the same way that the native tabs respond differently to the same button).

To make this possible a signal is sent from the GUI (at the time of writing, only Touchy) to the embedded tab. The signal is of type "Gladevcp" and the two messages sent are "Visible" and "Hidden". (Note that the signals have a fixed length of 20 characters so only the first characters should be used in any comparison, hence the [:7] below.) A sample handler for these signals is:

```
# This catches our messages from another program
def event(self,w,event):
    print(event.message_type,event.data)
    if event.message_type == 'Gladevcp':
        if event.data[:7] == 'Visible':
            self.active = True
        else:
            self.active = False

# connect to client-events from the host GUI
def on_map_event(self, widget, data=None):
    top = widget.get_toplevel()
    print("map event")
    top.connect('client-event', self.event)
```

If during module inspection GladeVCP finds a function `get_handlers`, it calls it as follows:

```
get_handlers(halcomp,builder,useropts)
```

The arguments are:

- `halcomp` - refers to the HAL component under construction
- `builder` - widget tree - result of reading the UI definition (either referring to a `GtkBuilder` or `libglade`-type object)
- `useropts` - a list of strings collected from the GladeVCP command line `-U <useropts>` option

GladeVCP then inspects the list of class instances and retrieves their method names. Qualifying method names are connected to the widget tree as signal handlers. Only method names which do not begin with an `_` (underscore) are considered.

Note that regardless whether you're using the `libglade` or the new `GtkBuilder` format for your Glade UI, widgets can always be referred to as `builder.get_object(<widgetname>)`. Also, the complete list of widgets is available as `builder.get_objects()` regardless of UI format.

10.3.8.6 Initialization sequence

It is important to know in which state of affairs your `get_handlers()` function is called so you know what is safe to do there and what not. First, modules are imported and initialized in command line order. After successful import, `get_handlers()` is called in the following state:

- The widget tree is created, but not yet realized (no `toplevel.window.show()` has been executed yet).
- The `halcomp` HAL component is set up and all HAL widget's pins have already been added to it.
- It is safe to add more HAL pins because `halcomp.ready()` has not yet been called at this point, so you may add your own pins, for instance in the class `__init__()` method.

Once all modules have been imported and method names extracted, the following steps happen:

- All qualifying method names will be connected to the widget tree with `connect_signals()/signal_autoc` (depending on the type of UI imported - `GtkBuilder` vs the old `libglade` format).
- The HAL component is finalized with `halcomp.ready()`.
- If a window ID was passed as argument, the widget tree is re-parented to run in this window, and Glade's `toplevel.window1` is abandoned (see FAQ).
- If a HAL command file was passed with `-H halfile`, it is executed with `halcmd`.
- The Gtk main loop is run.

So when your handler class is initialized, all widgets are existent but not yet realized (displayed on screen). And the HAL component isn't ready as well, so it's unsafe to access pins values in your `__init__()` method.

If you want to have a callback to execute at program start after it is safe to access HAL pins, then a connect a handler to the `realize` signal of the top level `window1` (which might be its only real purpose). At this point GladeVCP is done with all setup tasks, the HAL file has been run, and GladeVCP is about to enter the Gtk main loop.

10.3.8.7 Multiple callbacks with the same name

Within a class, method names must be unique. However, it is OK to have multiple class instances passed to GladeVCP by `get_handlers()` with identically named methods. When the corresponding signal occurs, these methods will be called in definition order - module by module, and within a module, in the order class instances are returned by `get_handlers()`.

10.3.8.8 The GladeVCP -U <useropts> flag

Instead of extending GladeVCP for any conceivable option which could potentially be useful for a handler class, you may use the `-U <useroption>` flag (repeatedly if you wish). This flag collects a list of `<useroption>` strings. This list is passed to the `get_handlers()` function (`useropts` argument). Your code is free to interpret these strings as you see fit. An possible usage would be to pass them to the Python `exec` function in your `get_handlers()` as follows:

```
debug = 0
...
def get_handlers(halcomp,builder,useropts):
    ...
    global debug # assuming there's a global var
    for cmd in useropts:
        exec cmd in globals()
```

This way you can pass arbitrary Python statements to your module through the `gladevc -U` option, for example:

```
gladevc -U debug=42 -U "print 'debug=%d' % debug" ...
```

This should set `debug` to 2 and confirm that your module actually did it.

10.3.8.9 Persistent variables in GladeVCP

An annoying aspect of GladeVCP in its earlier form and PyVCP is the fact that you may change values and HAL pins through text entry, sliders, spin boxes, toggle buttons etc, but their settings are not saved and restored at the next run of LinuxCNC - they start at the default value as set in the panel or widget definition.

GladeVCP has an easy-to-use mechanism to save and restore the state of HAL widgets, and program variables (in fact any instance attribute of type `int`, `float`, `bool` or `string`).

This mechanism uses the popular INI file format to save and reload persistent attributes.

Persistence, program versions and the signature check Imagine renaming, adding or deleting widgets in Glade: an `.INI` file lying around from a previous program version, or an entirely different user interface, would be not be able to restore the state properly since variables and types might have changed.

GladeVCP detects this situation by a signature which depends on all object names and types which are saved and to be restored. In the case of signature mismatch, a new `INI` file with default settings is generated.

10.3.8.10 Using persistent variables

If you want any of Gtk widget state, HAL widgets output pin's values and/or class attributes of your handler class to be retained across invocations, proceed as follows:

- Import the `gladevc.persistence` module.

- Decide which instance attributes, and their default values you want to have retained, if any.
- Decide which widgets should have their state retained.
- Describe these decisions in your handler class' *init()* method through a nested dictionary as follows:

```
def __init__(self, halcomp, builder, useropts):
    self.halcomp = halcomp
    self.builder = builder
    self.useropts = useropts
    self.defaults = {
        # the following names will be saved/restored as method attributes
        # the save/restore mechanism is strongly typed - the variables type will be derived ←
        # from the type of the
        # initialization value. Currently supported types are: int, float, bool, string
        IniFile.vars : { 'nhits' : 0, 'a': 1.67, 'd': True, 'c' : "a string"},
        # to save/restore all widget's state which might remotely make sense, add this:
        IniFile.widgets : widget_defaults(builder.get_objects())
        # a sensible alternative might be to retain only all HAL output widgets' state:
        # IniFile.widgets: widget_defaults(select_widgets(self.builder.get_objects(), ←
        #                               hal_only=True, output_only = True)),
    }
```

Then associate an INI file with this descriptor:

```
self.ini_filename = __name__ + '.ini'
self.ini = IniFile(self.ini_filename, self.defaults, self.builder)
self.ini.restore_state(self)
```

After *restore_state()*, *self* will have attributes set if as running the following:

```
self.nhits = 0
self.a = 1.67
self.d = True
self.c = "a string"
```

Note that types are saved and preserved on restore. This example assumes that the INI file didn't exist or had the default values from *self.defaults*.

After this incantation, you can use the following *IniFile* methods:

ini.save_state(obj)

Saves *obj*'s attributes as per *IniFile.vars* dictionary and the widget state as described in *IniFile.widgets* in *self.defaults*.

ini.create_default_ini()

Create an INI file with default values.

ini.restore_state(obj)

Restore HAL out pins and *obj*'s attributes as saved/initialized to default as above.

10.3.8.11 Saving the state on GladeVCP shutdown

To save the widget and/or variable state on exit, proceed as follows:

- Select some interior widget (type is not important, for instance a table).
- In the *Signals* tab, select *GtkObject*. It should show a *destroy* signal in the first column.

- Add the handler name, e.g. `on_destroy` to the second column.
- Add a Python handler like below:

```
import gtk
...
def on_destroy(self, obj, data=None):
    self.ini.save_state(self)
```

This will save state and shutdown GladeVCP properly, regardless whether the panel is embedded in AXIS, or a standalone window.



Caution

Do not use `window1` (the toplevel window) to connect a destroy event. Due to the way a GladeVCP panel interacts with AXIS if a panel is embedded within AXIS, **window1 will not receive destroy events properly**. However, since on shutdown all widgets are destroyed, anyone will do. Recommended: use a second-level widget - for instance, if you have a table container in your panel, use that.

Next time you start the GladeVCP application, the widgets should come up in the state when the application was closed.



Caution

The `GtkWidget` line has a similarly sounding *destroy-event* - **dont use that to connect to the `on_destroy` handler, it wont work** - make sure you use the *destroy* event from the `GtkObject` line.

10.3.8.12 Saving state when Ctrl-C is pressed

By default, the reaction of GladeVCP to a Ctrl-C event is to just exit - without saving state. To make sure that this case is covered, add a handler call `on_unix_signal` which will be automatically be called on Ctrl-C (actually on the SIGINT and SIGTERM signals). Example

```
def on_unix_signal(self, signum, stack_frame):
    print("on_unix_signal(): signal %d received, saving state" % (signum))
    self.ini.save_state(self)
```

10.3.8.13 Hand-editing INI (.ini) files

You can do that, but note that the values in `self.defaults` override your edits if there is a syntax or type error in your edit. The error is detected, a console message will hint about that happened, and the bad inifile will be renamed to have the `.BAD` suffix. Subsequent bad INI files overwrite earlier `.BAD` files.

10.3.8.14 Adding HAL pins

If you need HAL pins which are not associated with a specific HAL widget, add them as follows:

```
import hal_glib
...
# in your handler class __init__():
self.example_trigger = hal_glib.GPin(halcomp.newpin('example-trigger', hal.HAL_BIT, hal. ←
    HAL_IN))
```


To get a callback when this pin's value changes, associate a value-change callback with this pin, add:

```
self.example_trigger.connect('value-changed', self._on_example_trigger_change)
```

and define a callback method (or function, in this case leave out the self parameter):

```
# note '_' - this method will not be visible to the widget tree
def _on_example_trigger_change(self, pin, userdata=None):
    print("pin value changed to:" % (pin.get()))
```

10.3.8.15 Adding timers

Since GladeVCP uses Gtk widgets which rely on the GObject base class, the full glib functionality is available. Here is an example for a timer callback:

```
def _on_timer_tick(self, userdata=None):
    ...
    return True # to restart the timer; return False for on-shot
    ...
# demonstrate a slow background timer - granularity is one second
# for a faster timer (granularity 1 ms), use this:
# glib.timeout_add(100, self._on_timer_tick, userdata) # 10Hz
glib.timeout_add_seconds(1, self._on_timer_tick)
```

10.3.8.16 Setting HAL widget properties programmatically

With glade, widget properties are typically set fixed while editing. You can, however, set widget properties at runtime, for instance from INI file values, which would typically be done in the handler initialization code. Setting properties from HAL pin values is possible, too.

In the following example (assuming a HAL Meter widget called meter), the meter's min value is set from an INI file parameter at startup, and the max value is set via a HAL pin, which causes the widget's scale to readjust dynamically:

```
import linuxcnc
import os
import hal
import hal_glib

class HandlerClass:

    def _on_max_value_change(self, hal_pin, data=None):
        self.meter.max = float(hal_pin.get())
        self.meter.queue_draw() # force a widget redraw

    def __init__(self, halcomp, builder, useropts):
        self.builder = builder

        # HAL pin with change callback.
        # When the pin's value changes the callback is executed.
        self.max_value = hal_glib.GPin(halcomp.newpin('max-value', hal.HAL_FLOAT, hal. ↵
            HAL_IN))
        self.max_value.connect('value-changed', self._on_max_value_change)

    inifile = linuxcnc.ini(os.getenv("INI_FILE_NAME"))
    mmin = float(inifile.find("METER", "MIN") or 0.0)
    self.meter = self.builder.get_object('meter')
    self.meter.min = mmin
```

```
def get_handlers(halcomp, builder, useropts):
    return [HandlerClass(halcomp, builder, useropts)]
```

10.3.8.17 Examples, and rolling your own GladeVCP application

Visit `linuxcnc_root_directory/configs/apps/gladevcp` for running examples and starters for your own projects.

10.3.9 FAQ

1. *I get an unexpected unmap event in my handler function right after startup. What's this?*

This is a consequence of your Glade UI file having the `window1 Visible` property set to `True`, together with re-parenting the GladeVCP window into `AXIS` or `touchy`. The GladeVCP widget tree is created, including a top level window, and then *reparented into AXIS*, leaving that toplevel window laying around orphaned. To avoid having this useless empty window hanging around, it is unmapped (made invisible), which is the cause of the `unmap` signal you get. Suggested fix: set `window1.visible` to `False`, and ignore an initial `unmap` event.

2. *My GladeVCP program starts, but no window appears where I expect it to be?*

The window `AXIS` allocates for GladeVCP will obtain the *natural size* of all its child widgets combined. It's the child widget's job to request a size (width and/or height). However, not all widgets do request a width greater than 0, for instance the `Graph` widget in its current form. If there's such a widget in your Glade file and it's the one which defines the layout you might want to set its width explicitly. Note that setting the `window1 width` and `height` properties in Glade does not make sense because this window will be orphaned during re-parenting and hence its geometry will have no impact on layout (see above). The general rule is: if you manually run a UI file with `gladevcp <uifile>` and its window has reasonable geometry, it should come up in `AXIS` properly as well.

3. *I want a blinking LED, but it wont blink*

I ticked the checkbox to let it blink with 100 msec interval. It wont blink, and I get a startup warning: `Warning: value "0" of type 'gint' is invalid or out of range for property 'led-blink-rate' of type 'gint'?` This seems to be a glade bug. Just type over the blink rate field, and save again - this works for me.

4. *My GladeVCP panel in AXIS doesn't save state when I close AXIS, although I defined an on_destroy handler linked to the window destroy signal*

Very likely this handler is linked to `window1`, which due to reparenting isn't usable for this purpose. Please link the `on_destroy` handler to the `destroy` signal of an interior window. For instance, I have a `notebook` inside `window1`, and linked `on_destroy` to the `notebooks destroy` signal, and that works fine. It doesn't work for `window1`.

5. *I want to set the background color or text of a HAL_Label widget depending on its HAL pin value*

See the example in `configs/apps/gladevcp/colored-label`. Setting the background color of a `GtkLabel` widget (and `HAL_Label` is derived from `GtkLabel`) is a bit tricky. The `GtkLabel` widget has no window object of its own for performance reasons, and only window objects can have a background color. The solution is to enclose the `Label` in an `EventBox` container, which has a window but is otherwise invisible - see the `coloredlabel.ui` file.

6. *I defined a hal_spinbutton widget in glade, and set a default value property in the corresponding adjustment. It comes up with zero?*

this is due to a bug in the old Gtk version distributed with Ubuntu 8.04 and 10.04, and is likely to be the case for all widgets using adjustment. The workaround mentioned for instance in <http://osdir.com/ml/gtk-app-devel-list/2010-04/msg00129.html> does not reliably set the HAL pin value, it is better to set it explicitly in an `on_realize` signal handler during widget creation. See the example in `configs/apps/gladevcp/by-widget/spinbutton.{ui,py}`.

10.3.10 Troubleshooting

- make sure you have the development version of LinuxCNC installed. You don't need the `axisrc` file any more, this was mentioned in the old GladeVCP wiki page.
- run GladeVCP or AXIS from a terminal window. If you get Python errors, check whether there's still a `/usr/lib/python2.6/dist-packages/hal.so` file lying around besides the newer `/usr/lib/python2.6` (note underscore); if yes, remove the `hal.so` file. It has been superseded by `hal.py` in the same directory and confuses the import mechanism.
- if you're using `run-in-place`, do a *make clean* to remove any accidentally left over `hal.so` file, then *make*.
- if you're using `HAL_table` or `HAL_HBox` widgets, be aware they have an HAL pin associated with it which is off by default. This pin controls whether these container's children are active or not.

10.3.11 Implementation note: Key handling in AXIS

We believe key handling works OK, but since it is new code, we're telling about it you so you can watch out for problems; please let us know of errors or odd behavior. This is the story:

AXIS uses the TkInter widget set. GladeVCP applications use Gtk widgets and run in a separate process context. They are hooked into AXIS with the Xembed protocol. This allows a child application like GladeVCP to properly fit in a parent's window, and - in theory - have integrated event handling.

However, this assumes that both parent and child application properly support the Xembed protocol, which Gtk does, but TkInter doesn't. A consequence of this is that certain keys would not be forwarded from a GladeVCP panel to AXIS properly under all circumstances. One of these situations was the case when an Entry, or SpinButton widget had focus: in this case, for instance an Escape key would not have been forwarded to AXIS and cause an abort as it should, with potentially disastrous consequences.

Therefore, key events in GladeVCP are explicitly handled, and selectively forwarded to AXIS, to assure that such situations cannot arise. For details, see the `keyboard_forward()` function in `lib/python/gladevcp`.

10.3.12 Adding Custom Widgets

The LinuxCNC Wiki has information on adding custom widgets to GladeVCP. [GladeVCP Custom Widgets](#)

10.3.13 Auxiliary GladeVCP Applications

Support is provided for independently installed GladeVCP applications that conform to system directory placements as defined by the `LINUXCNC_AUX_GLADEVCP` and `LINUXCNC_AUX_EXAMPLES` items reported by the script `linuxcnc_var`:

```
$ linuxcnc_var LINUXCNC_AUX_GLADEVCP
/usr/share/linuxcnc/aux_gladevcp
$ linuxcnc_var LINUXCNC_AUX_EXAMPLES
/usr/share/linuxcnc/aux_examples
```

The system directory defined by `LINUXCNC_AUX_GLADEVCP` (`/usr/share/linuxcnc/aux_gladevcp`) specifies the location for a GladeVCP-compatible python file(s) and related subdirectories. The python file is imported at GladeVCP startup and made available to subsequent GladeVCP applications including embedded usage in supporting GUIs.

The system directory defined by `LINUXCNC_AUX_EXAMPLES` (`/usr/share/linuxcnc/aux_examples`) specifies the location of example configuration subdirectories used for auxiliary applications. See the getting-started/running-linuxcnc section for *Adding Configuration Selection Items*.

For testing, a runtime specification of auxiliary applications may be specified using the exported environmental variable: `GLADEVCP_EXTRAS`. This variable should be a path list of one or more configuration directories separated by a (`:`). Typically, this variable would be set in a shell starting linuxcnc or in a user's `~/.profile` startup script. Example:

```
export GLADEVCP_EXTRAS=~/mygladevcp:/opt/othergladevcp
```

Files found in directories specified with the environmental variable `GLADEVCP_EXTRAS` supersede identically-named files within subdirectories of the system directory specified by `LINUXCNC_AUX_GLADEVCP` (e.g., `/usr/share/linuxcnc/aux_gladevcp`). This provision allows a developer to test an application by exporting `GLADEVCP_EXTRAS` to specify a private application directory without removing a system-installed application directory. Messages indicating rejected duplicates are printed to stdout.

Note

Support for auxiliary GladeVCP applications requires a python module named *importlib*. This module may not be available in older installations like Ubuntu-Lucid.

10.4 Qtvcp

Qtvcp is an infrastructure to display a custom CNC screen or control panel in LinuxCNC. It displays a UI file built with the QTDesigner screen editor or combines this with python programming to create a GUI screen for running a CNC machine. Qtvcp is completely customizable - you can add different buttons and status LEDs etc. or add python code for even finer grain customization.

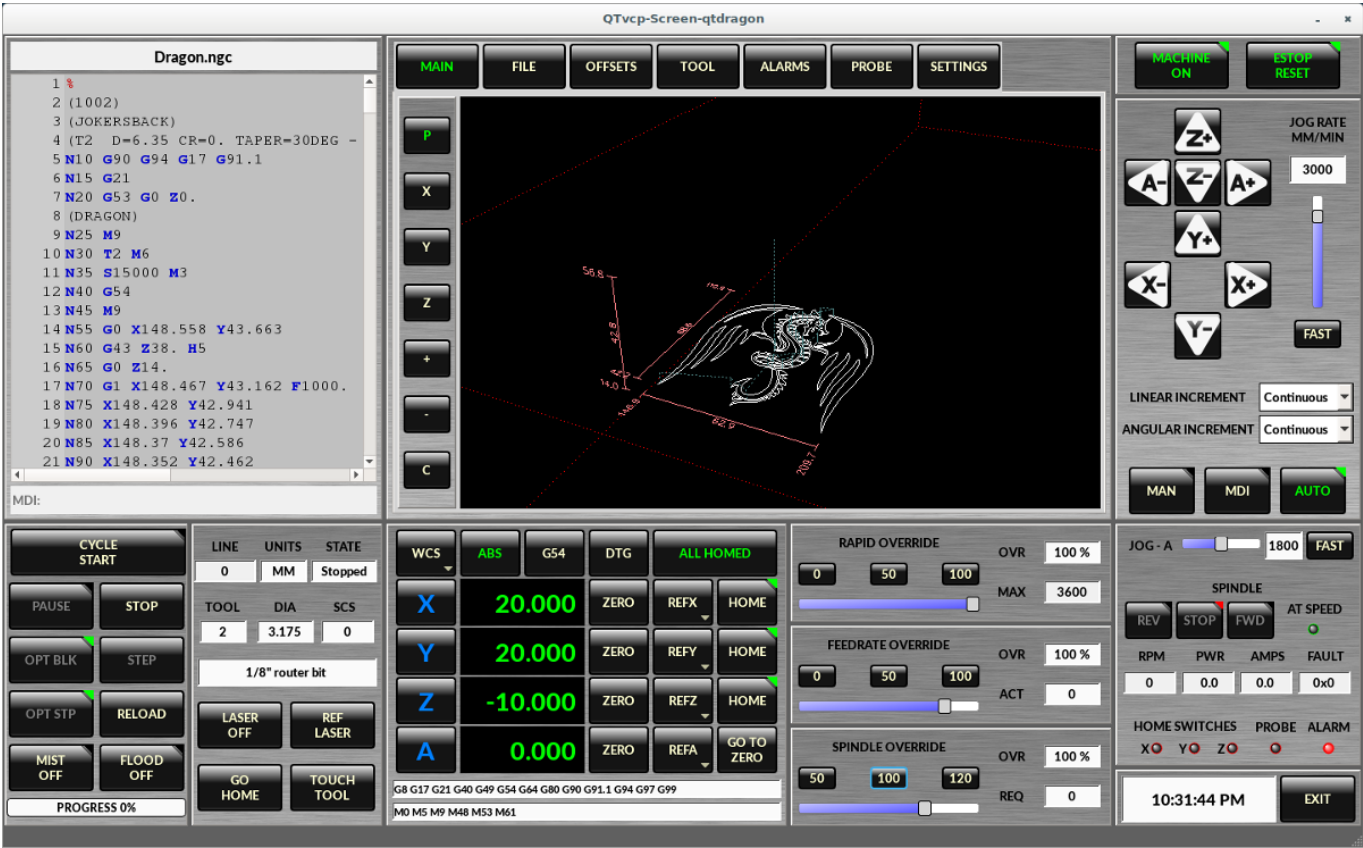


Figure 10.55: qtdragon - 3 or 4 Axis Sample

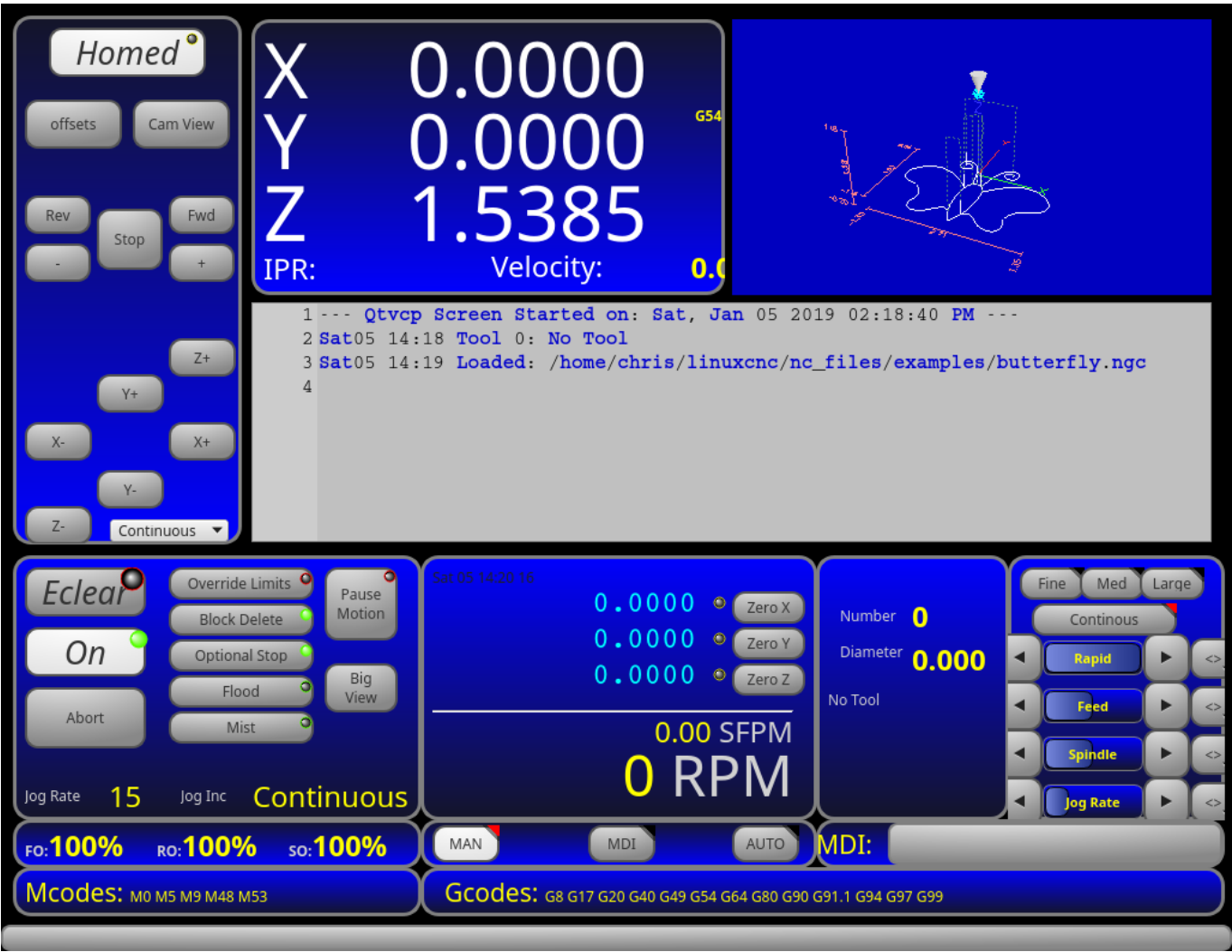


Figure 10.56: qtdefault - 3 Axis Sample

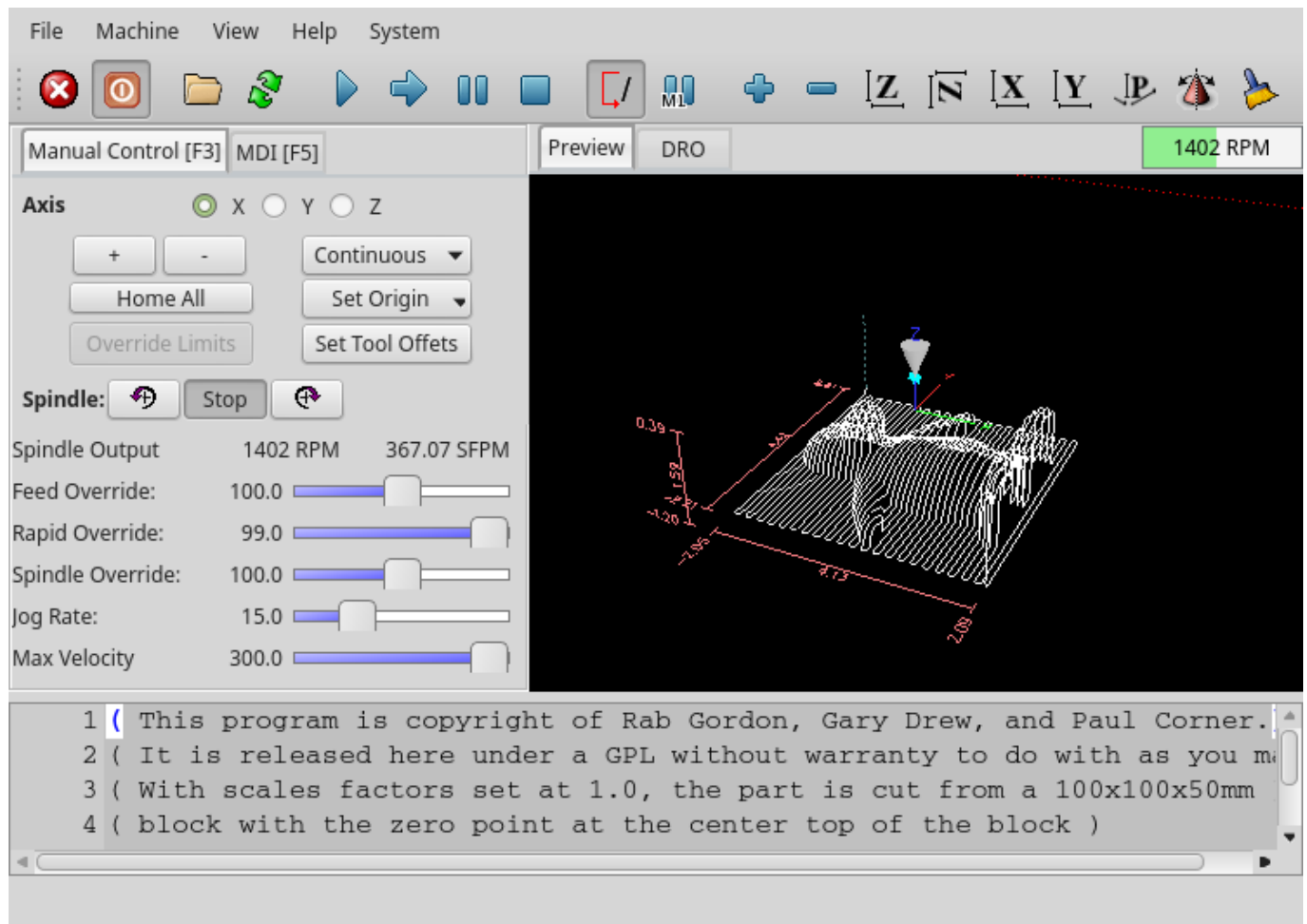


Figure 10.57: Qtaxis - Self Adjusting Axis Sample

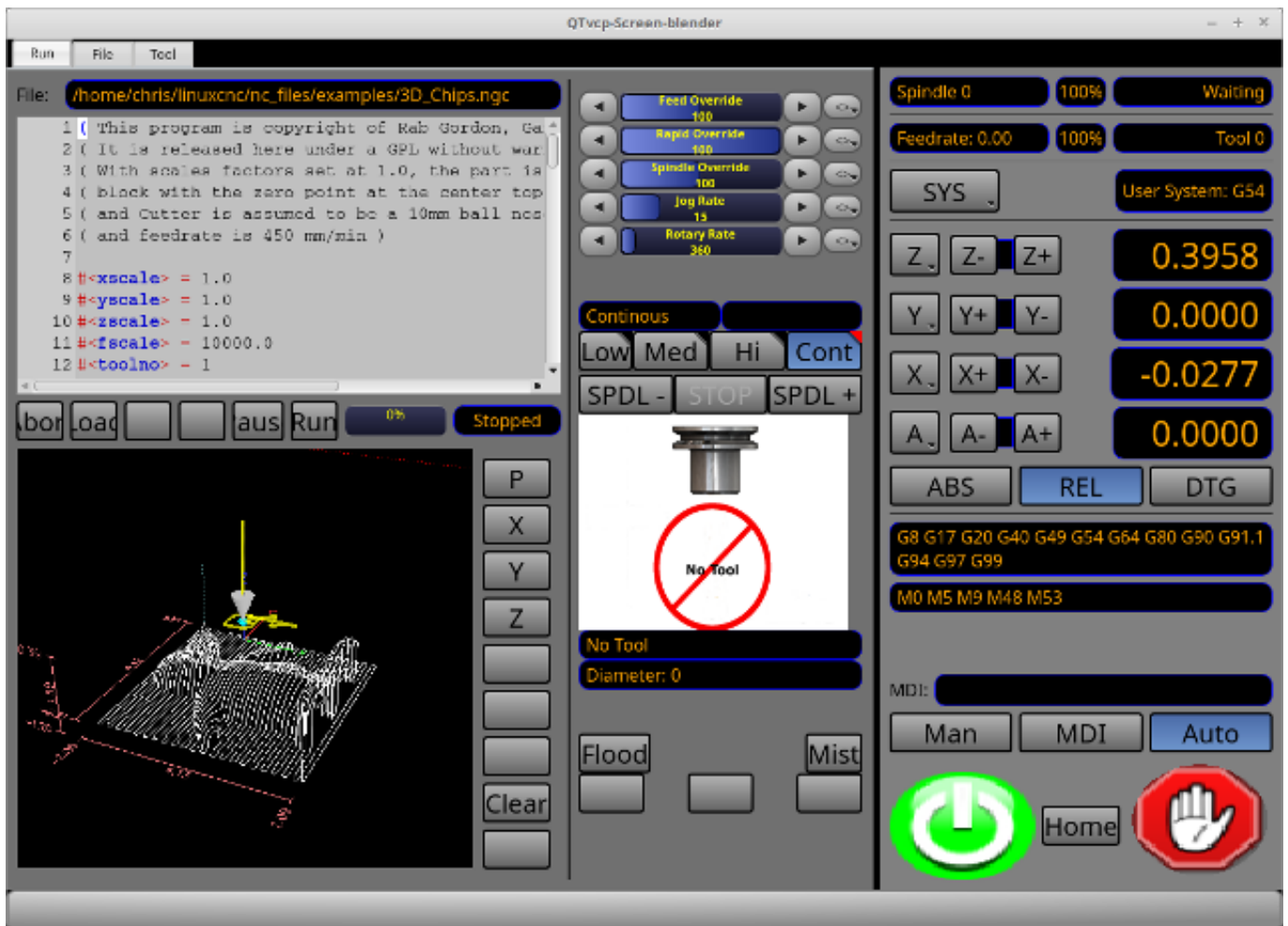


Figure 10.58: Blender - 4 Axis Sample



Figure 10.59: X1mill - 4 Axis Sample



Figure 10.60: cam_align - Camera alignment VCP

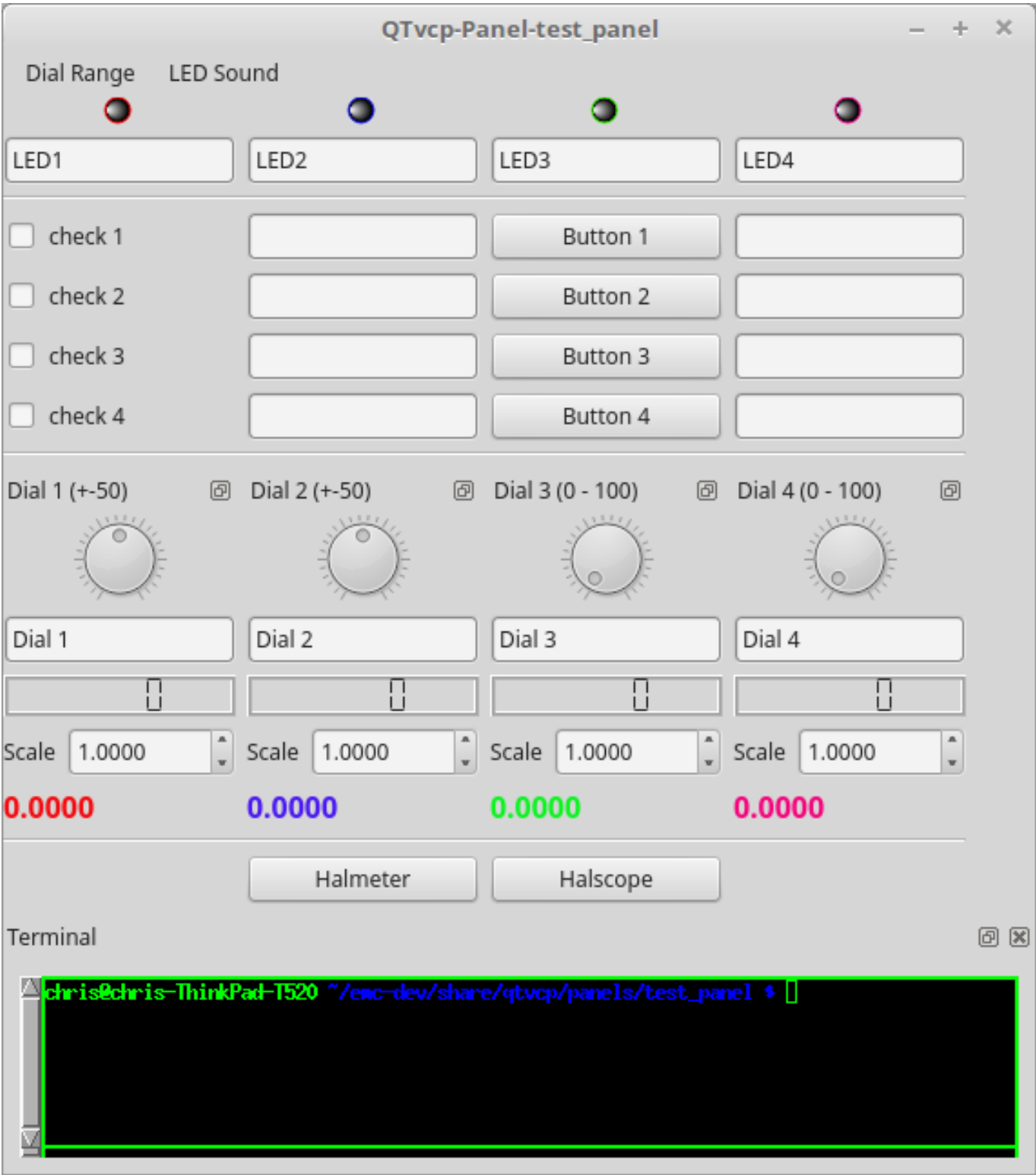


Figure 10.61: test panel - Test Panel VCP

10.4.1 Overview

There are two files that can be used, individually or in combination to add customization. A UI file that is made with QT's Designer graphical editor. A handler file which is a text file with python code. Normally qtvcp uses the stock UI and handler file. You can specify qtvcp to use *local* UI and handler files. A *local* file is one that is in the configuration folder that defines the rest of the machine's requirements. One is not restricted to adding a custom panel on the right or a custom tab. qtvcp leverages *QT Designer* (the editor) and *PyQT5* (the widget toolkit). QTvcp has some special widgets and actions added just for LinuxCNC. There are special widgets to bridge third party widgets to HAL pins. It's possible to create widget responses by connecting signals to python code in the handler file.

10.4.1.1 Qtvcp Widgets

Qtvcp uses the PyQt5 toolkit's widgets for linuxcnc integration. Widget is the general name for objects such as buttons and labels in PyQt5. You are free to use any available widgets in the QTDesigner editor. There are also special widgets made for linuxcnc that make integration easier. These are split in three heading on the left side of the editor. One is for HAL only widgets. One is for cnc control widgets. One is for dialog widgets. you are free to mix them in any way on your panel. A very important widget for CNC control is the screenoptions widget. It does not add anything visually to the screen. But allows important details to be selected rather than be coded in the handler file.

10.4.1.2 INI Settings

If you are using this to make a CNC control screen: Under the [DISPLAY] heading:

```
DISPLAY = qtvcp <options> <screen_name>
options:
  -a set window always on top
  -c HAL component name. Default is to use the UI file name.
  -d debugging on
  -f fullscreen the window
  -g geometry: WIDTHxHEIGHT+XOFFSET+YOFFSET example: -g 200x400+0+100
  -H execute HAL statements from FILE with halcmd after the component is set up and ready
  -i set logging level to INFO
  -m maximize window
  -o pass a string to qtvcp's handler file under self.w.USEROPTIONS_ list variable. can be multiple -o
  -t theme. Default is system theme
  -u file path of a substitute handler file
  -v enable VERBOSE debug output
  -x embed into a X11 window that doesn't support embedding.
  --push_xid send qtvcp's X11 window id number to standard output; for embedding

all <options> must be before <screen name>
<screen_name> is the base name of the .ui and _handler.py files.
If <screen_name> is missing the default screen will be loaded.
```

Qtvcp assumes the UI file and the handler file use this same base name. Qtvcp will search the LinuxCNC configuration file that was launched first for the files, then in the system skin folder. (The skin folders holds standard screens.)

```
[DISPLAY]
CYCLE_TIME = 100
GRAPHICS_CYCLE_TIME = 100
HALPIN_CYCLE = 100
```

Adjust the response rate of the GUI updates in milli seconds. Defaults to 100, useable range 50 - 200 The widgets, graphics and HAL pin update can be set separately. If the update time is not set right the screen can become unresponsive or very jerky.

10.4.1.3 QTDesigner UI File

A designer file is a text file organized in the XML standard that describes the layout and the widgets of the screen. PyQt5 uses this file to build the display and react to those widgets. The QTDesigner editor makes it relatively easy to build and edit this file.

10.4.1.4 Handler Files

A handler file is a file containing python code, which qtvcp adds to it's default routines. A handler file allows one to modify defaults, or add logic to a qtvcp skin without having to modify qtvcp's core code. In this way you can have custom behaviour. If present a handler file will be loaded. Only one file is allowed.

10.4.1.5 Libraries modules

Qtvcp as built does little more then display the screen and react to widgets. For more prebuilt behaviours there are available libraries. (found in lib/python/qtvcp/lib in RIP linuxcnc install) libraries are prebuilt python modules that give added features to Qtvcp. In this way you can select what features you want - yet don't have to build common ones yourself. Such libraries include:

- audio_player
- aux_program_loader
- keybindings
- message
- preferences
- notify
- virtual_keyboard
- machine_log

10.4.1.6 Themes

Themes are a way to modify the look and feel of the widgets on the screen. For instance the color or size of buttons and sliders can be changed using themes. The Windows theme is default for screens. System theme is default for panels. to see available themes load qtvcp with -d -t SHOWTHEMES .

qtvcp can also be customized with Qt stylesheets using css.

10.4.1.7 Local Files

If present, local UI files in the configuration folder will be loaded instead of the stock UI files. Local UI files allow you to use your customized designs rather then the default screens. QTVCP will look for a folder name MYNAME (in the launched configuration folder that holds the INI file). In that folder QTVCP will load any of the available files; MYNAME.ui, MYNAME_handler.py and MYNAME.qss.

10.4.1.8 Modifying Stock Screens

There are three ways to customize a screen/panel.

Minor StyleSheet changes: StyleSheets can be used to set Qt properties. If a widget uses properties they usually can be modified by stylesheets. ie:

```
State_LED #name_0f_led{
qproperty-color: red;
qproperty-diameter: 20;
qproperty-flashRate: 150;
}
```

Minor python code changes: A file can be added to add commands to the screen, after the handlerfile is parsed. In the INI file under the [DISPLAY] heading add `USER_COMMAND_FILE = PATH` `PATH` can be any valid path, it can use `~` for home directory or `WORKINGDIRECTORY` or `CONFIGDIRECTORY` to represent Qtvcp's idea of those directories. ie:

```
[DISPLAY]
USER_COMMAND_FILE = CONFIGDIRECTORY/qtdragon_added_commands
```

If no entry is found in the INI, Qtvcp will look in the default path. The default path is in the configuration directory as a hidden file using the screen basename and rc. ie: `CONFIGDIRECTORY/.qtdragonrc`

This file will be read and executed as python code in context of the handler file. Only local functions and local attributes can be referenced. Global libraries can not be referenced. (usual seen as all capital words with no preceding self.) What can be used can vary by screen and development cycle.

valid example:

```
self.w.setWindowTitle('My Title Test')
```

Full creative control: If you wish to modify a stock screen with full control, copy it's UI and handler file to your configuration folder. There is a QtVCP panel to help with this. Open a terminal and type `qtvcp copy_dialog` and a dialog will show to select the screen and destination folder. This will copy all the file - delete the ones you don't wish to modify so that the original files will be used. If you wish to name your screen differently then the builtin screen's default name - change the basename in the edit box.

10.4.2 VCP Panels

Qtvcp can be used to create control panels that interface with HAL.

10.4.2.1 Builtin panels

There are several builtin HAL panels available. in a terminal type `qtvcp <return>` to see a list.

- `test_panel` - collect of useful widgets for testing HAL component. Including speech of LED state.
- `cam_align` - a camera display widget for rotational alignment
- `sim_panel` - a small control panel to simulate MPG jogging controls etc. for simulated configurations
- `vismach_mill_xyz` - 3d openGL view of a 3 axis milling machine



Figure 10.62: qtvismach- 3 Axis Mill Builtin panel

```
loadusr qtvcp test_panel
```

You can of course make your own panel and load it. If you made a ui file named *my_panel.ui* and name the following HAL file, *my_panel.hal* You would then load this from a terminal with `halrun -I -f my_panel.hal`

```
# load realtime components
loadrt threads
loadrt classicladder_rt

# load user space programs
loadusr classicladder
loadusr -Wn my_panel qtvcp my_panel.ui

# add components to thread
addf classicladder.0.refresh thread1
```

```
# connect pins
net bit-input1      test_panel.checkbox_1      classicladder.0.in-00
net bit-hide        test_panel.checkbox_4      classicladder.0.hide_gui

net bit-output1     test_panel.led_1           classicladder.0.out-00

net s32-in1         test_panel.doublescale_1-s classicladder.0.s32in-00

# start thread
start
```

In this case we load qtvcp using -Wn; which waits for the panel to finish loading before continuing to run the next HAL command. This is so the HAL pins from the panel are finished in case they are used in the rest of the file.

10.4.3 Build a simple clean-sheet custom screen

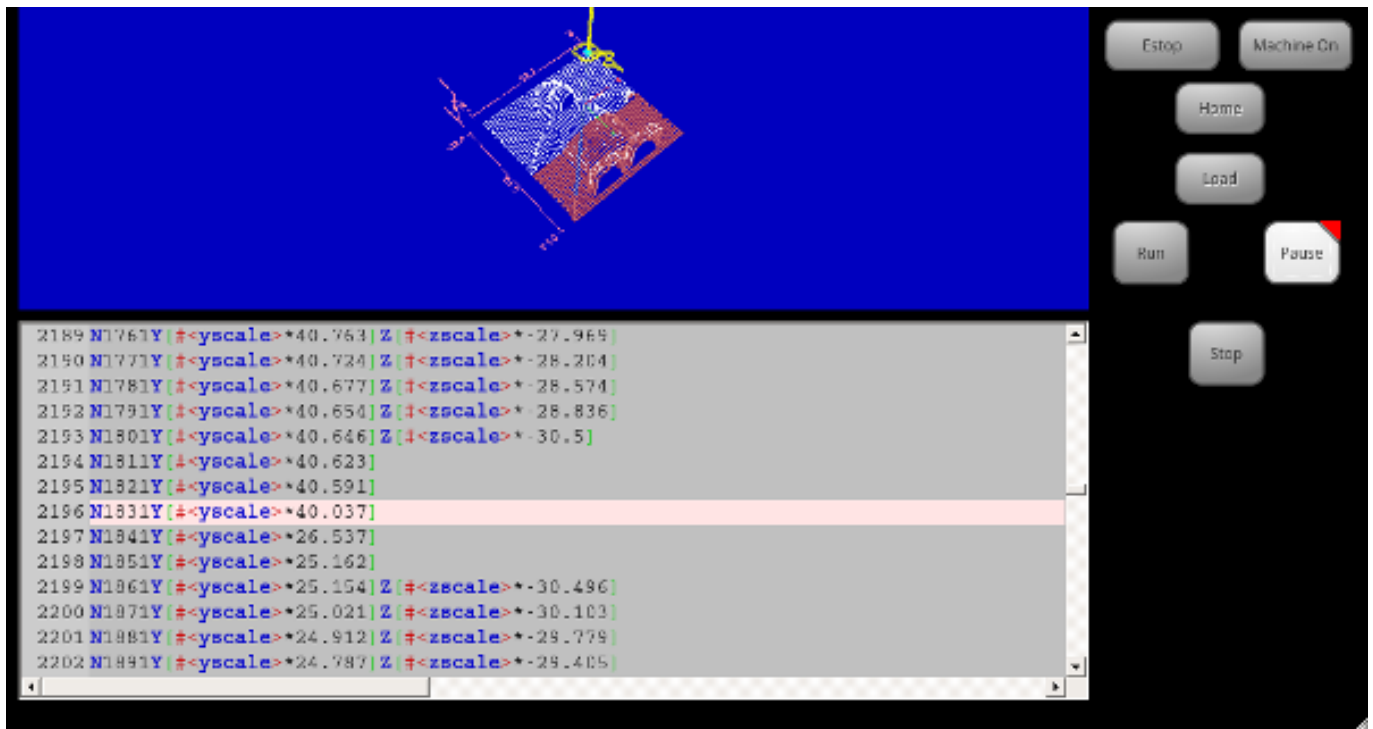


Figure 10.63: Ugly custom screen

10.4.3.1 Overview

To build a panel or screen use QTDesigner to build a design you like. Save this design to your configuration folder with a name of your choice, ending with .ui modify the configurations INI file to load qtvcp with your new .ui file. Then connect any required HAL pins in a HAL file

10.4.3.2 Get Designer to include linuxcnc widgets

You must have designer installed; These commands should add it: Or use your package manager to install the same:

```
sudo apt-get install qttools5-dev-tools qttools5-dev libpython3-dev
```

Then you must add a link to the qtvcp_plugin.py to the folder that designer will search.

In a RIP version of linuxcnc qtvcp_plugin.py will be in:

```
'~/LINUXCNC_PROJECT_NAME/lib/python/qtvcpl/plugins/qtvcpl_plugin.py'
```

Package installed version should be:

```
'usr/lib/python2.7/qtvcpl/plugins/qtvcpl_plugin.py' or  
'usr/lib/python2.7/dist-packages/qtvcpl/plugins/qtvcpl_plugin.py'
```

Make a link file to the above file and move it to one of the places Designer searches in:

Designer searches in these two place for links (pick one): This can be:

```
'/usr/lib/x86_64-linux-gnu/qt5/plugins/designer/python' or  
'~/designer/plugins/python'
```

You may need to add the plugins/python folders

To start Designer:

for a RIP installed: open a terminal, set the environment for linuxcnc with the command: `. scripts/rip-environment` then load designer with : `designer -qt=5`

otherwise for an installed version, open a terminal and type `designer -qt=5`

If all goes right you will see the selectable linuxcnc widgets on the left hand side

10.4.3.3 build the screen .ui file

When Designer is first started there is a *New Form* dialog displayed. Pick *Main Window* and press the *create* button. Do not rename this window - Qtvcp requires the name to be *MainWindow*

A MainWindow widget is Displayed. Grab the corner of the window and resize to an appropriate size say 1000x600. right click on the window and click set minimum size. Do it again and set maximum size. Our sample widget will now not be resizable.

Drag and drop the screenoption widget onto the main window (anywhere). This widget doesn't add anything visually but sets up some common options. It's recommended to always add this widget before any other. Right click on the main window (not the screenoptions widget) and set the layout as vertical. The screenoption widget will now be fullsized.

On the right hand side there is a panel with tabs for a Property editor and an object inspector. On the Object inspector click on the screenoption. then switch to the property Editor. Under the heading *ScreenOptions* toggle *filedialog_option*.

Drag and drop a GCodeGraphics widget and a GcodeEditor widget. Place and resize them as you see fit leaving some room for buttons.

Now we will add action buttons. Add 7 action buttons on to the main window. If you double click the button, you can add text. Edit the button labels for *Estop*, *Machine On*, *Home*, *Load*, *Run*, *Pause* and *stop*. Action buttons default to no action so we must change the properties for defined functions. You can edit the properties directly in the property editor on the right side of designer. A convenient alternating is left double clicking on the button This will launch a Dialog that allows selecting actions while only display relevant data to the action.

We will describe the convenient way first:

- Right click the *Machine On* button and select *Set Actions*. When the Dialog displays, use the combobox to navigate to *MACHINE CONTROLS - Machine On*. In this case there is no option for this action so select ok. Now the button will turn the machine on when pressed

And now the direct way with Designer's property editor

- Select the *Machine On* button. Now go to the *Property Editor* on the right side of Designer. Scroll down until you find the *ActionButton* heading. You will see a list of properties and values. find the *machine on action* and click the checkbox. the button will now control machine on/off.

Do the same for all the other button with the addition of:

- With the *Home* button we must also change the *joint_number* property to -1, Which tells the controller to home all the axes rather than a specific axis.
 - With the *Pause* button under the heading *Indicated_PushButton* check the *indicator_option* and under the *QAbstractButton* heading check *checkable*
-

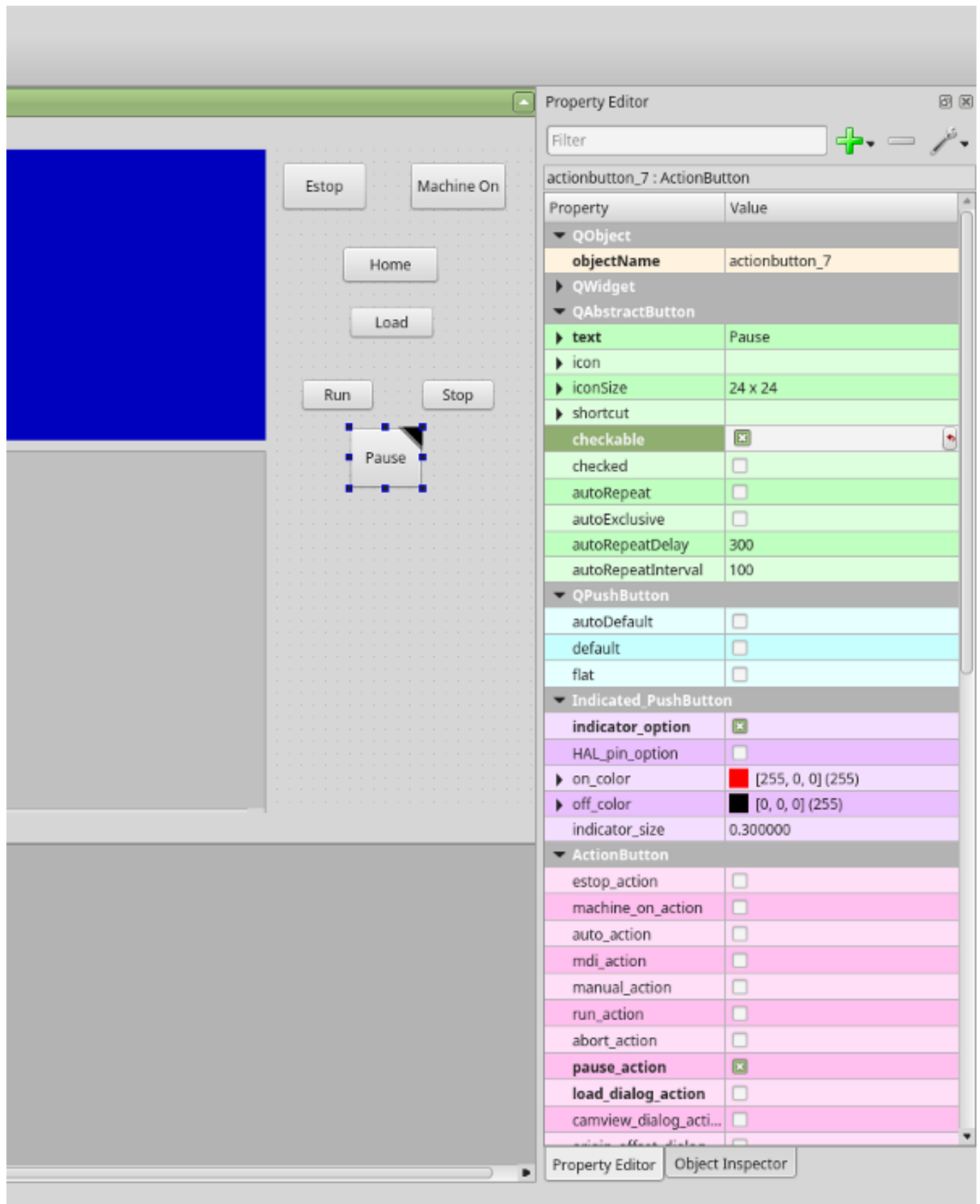


Figure 10.64: Qt Designer - Selecting Pause button's properties

We then need to save this design as *tester.ui* in the *sim/qtvc* folder We are saving it as *tester* as that is a file name that *qtvc* recognizes and will use a built in handler file to display it.

10.4.3.4 Handler file

a handler file is required. It allows customizations to be written in python. For instance keyboard controls are usually written in the handler file.

In this example the built in file *tester_handler.py* is automatically used. It does the minimum required to display the *tester.ui* defined screen and do basic keyboard jogging.

10.4.3.5 INI

If you are using *qtvc* to make a CNC control screen: Under the *[DISPLAY]* heading:

```
'DISPLAY = qtvc <screen_name>'
```

<screen_name> is the base name of the *.ui* and *_handler.py* files.

In our example there is already a sim configuration called *tester*, that we will use to display our test screen.

10.4.3.6 HAL

If your screen used widgets with HAL pins, then you must connect them in a HAL file. *Qtvc* looks in the INI file, under the heading *[HAL]* for the entry *POSTGUI_HALFILE=<filename>* Typically *<filename>* would be the screens base name *_postgui + .hal*

eg. *qtvc_postgui.hal*, but can be any legal filename. These commands are executed after the screen is built, guaranteeing the widget HAL pins are available. You can have multiple line of *POSTGUI_HALFILE=<filename>* in the INI. Each will be run one after the other in the order they appear.

Qtvc also looks in the INI file, under the heading *[HAL]* for the entry *POSTGUI_HALCMD=<command>* *<command>* would be any valid HAL command. These commands are executed after the screen is built, after all the *POSTGUI_HALFILEs* are run, guaranteeing the widget HAL pins are available. You can have multiple line of *POSTGUI_HALCMD=<command>* in the INI. Each will be run one after the other in the order they appear.

In our example there are no HAL pins to connect.

10.4.4 Handler file in detail

handler files are used to create custom controls using python.

10.4.4.1 Overview

Here is a sample handler file. It's broken up in sections for ease of discussion.

```
#####
# **** IMPORT SECTION **** #
#####
import sys
import os
import linuxcnc

from PyQt5 import QtCore, QtWidgets
```

```

from qtvcp.widgets.mdi_line import MDI_Line as MDI_WIDGET
from qtvcp.widgets.gcode_editor import GcodeEditor as GCODE
from qtvcp.lib.keybindings import Keylookup
from qtvcp.core import Status, Action

# Set up logging
from qtvcp import logger
LOG = logger.getLogger(__name__)

# Set the log level for this module
#LOG.setLevel(logger.INFO) # One of DEBUG, INFO, WARNING, ERROR, CRITICAL

#####
# **** INSTANTIATE LIBRARIES SECTION **** #
#####

KEYBIND = Keylookup()
STATUS = Status()
ACTION = Action()
#####
# **** HANDLER CLASS SECTION **** #
#####

class HandlerClass:

    #####
    # **** INITIALIZE **** #
    #####
    # widgets allows access to widgets from the qtvcp files
    # at this point the widgets and hal pins are not instantiated
    def __init__(self, halcomp, widgets, paths):
        self.hal = halcomp
        self.w = widgets
        self.PATHS = paths

    #####
    # SPECIAL FUNCTIONS SECTION #
    #####

    # at this point:
    # the widgets are instantiated.
    # the HAL pins are built but HAL is not set ready
    # This is where you make HAL pins or initialize state of widgets etc
    def initialized__(self):
        pass

    def processed_key_event__(self, receiver, event, is_pressed, key, code, shift, cntrl):
        # when typing in MDI, we don't want keybinding to call functions
        # so we catch and process the events directly.
        # We do want ESC, F1 and F2 to call keybinding functions though
        if code not in (QtCore.Qt.Key_Escape, QtCore.Qt.Key_F1, QtCore.Qt.Key_F2,
                        QtCore.Qt.Key_F3, QtCore.Qt.Key_F5, QtCore.Qt.Key_F5):

            # search for the top widget of whatever widget received the event
            # then check if it's one we want the keypress events to go to
            flag = False
            receiver2 = receiver
            while receiver2 is not None and not flag:
                if isinstance(receiver2, QtWidgets.QDialog):
                    flag = True
                    break

```

```

        if isinstance(receiver2, MDI_WIDGET):
            flag = True
            break
        if isinstance(receiver2, GCODE):
            flag = True
            break
        receiver2 = receiver2.parent()

    if flag:
        if isinstance(receiver2, GCODE):
            # if in manual do our keybindings - otherwise
            # send events to G-code widget
            if STATUS.is_man_mode() == False:
                if is_pressed:
                    receiver.keyPressEvent(event)
                    event.accept()
                    return True
            elif is_pressed:
                receiver.keyPressEvent(event)
                event.accept()
                return True
            else:
                event.accept()
                return True

    if event.isAutoRepeat(): return True

    # ok if we got here then try keybindings
    try:
        return KEYBIND.call(self,event,is_pressed,shift,ctrl)
    except NameError as e:
        LOG.debug('Exception in KEYBINDING: {}'.format (e))
    except Exception as e:
        LOG.debug('Exception in KEYBINDING:', exc_info=e)
        print('Error in, or no function for: %s in handler file for-%s'%(KEYBIND. ←
            convert(event),key))
        return False

#####
# CALLBACKS FROM STATUS #
#####

#####
# CALLBACKS FROM FORM #
#####

#####
# GENERAL FUNCTIONS #
#####

# keyboard jogging from key binding calls
# double the rate if fast is true
def kb_jog(self, state, joint, direction, fast = False, linear = True):
    if not STATUS.is_man_mode() or not STATUS.machine_is_on():
        return
    if linear:
        distance = STATUS.get_jog_increment()
        rate = STATUS.get_jograte()/60
    else:
        distance = STATUS.get_jog_increment_angular()
        rate = STATUS.get_jograte_angular()/60
    if state:

```

```

        if fast:
            rate = rate * 2
            ACTION.JOG(joint, direction, rate, distance)
        else:
            ACTION.JOG(joint, 0, 0, 0)

#####
# KEY BINDING CALLS #
#####

# Machine control
def on_keycall_ESTOP(self,event,state,shift,cntrl):
    if state:
        ACTION.SET_ESTOP_STATE(STATUS.estop_is_clear())
def on_keycall_POWER(self,event,state,shift,cntrl):
    if state:
        ACTION.SET_MACHINE_STATE(not STATUS.machine_is_on())
def on_keycall_HOME(self,event,state,shift,cntrl):
    if state:
        if STATUS.is_all_homed():
            ACTION.SET_MACHINE_UNHOMED(-1)
        else:
            ACTION.SET_MACHINE_HOMING(-1)
def on_keycall_ABORT(self,event,state,shift,cntrl):
    if state:
        if STATUS.stat.interp_state == linuxcnc.INTERP_IDLE:
            self.w.close()
        else:
            self.cmd.abort()

# Linear Jogging
def on_keycall_XPOS(self,event,state,shift,cntrl):
    self.kb_jog(state, 0, 1, shift)

def on_keycall_XNEG(self,event,state,shift,cntrl):
    self.kb_jog(state, 0, -1, shift)

def on_keycall_YPOS(self,event,state,shift,cntrl):
    self.kb_jog(state, 1, 1, shift)

def on_keycall_YNEG(self,event,state,shift,cntrl):
    self.kb_jog(state, 1, -1, shift)

def on_keycall_ZPOS(self,event,state,shift,cntrl):
    self.kb_jog(state, 2, 1, shift)

def on_keycall_ZNEG(self,event,state,shift,cntrl):
    self.kb_jog(state, 2, -1, shift)

def on_keycall_APOS(self,event,state,shift,cntrl):
    pass
    #self.kb_jog(state, 3, 1, shift, False)

def on_keycall_ANEG(self,event,state,shift,cntrl):
    pass
    #self.kb_jog(state, 3, -1, shift, linear=False)

#####
# *** closing event *** #
#####

#####

```

```

# required class boiler code #
#####

def __getitem__(self, item):
    return getattr(self, item)
def __setitem__(self, item, value):
    return setattr(self, item, value)

#####
# required handler boiler code #
#####

def get_handlers(halcomp,widgets,paths):
    return [HandlerClass(halcomp,widgets,paths)]

```

10.4.4.2 IMPORT SECTION

This section is for importing library modules required for your screen. It would be typical to import qtvcp's keybinding, Status and action libraries.

10.4.4.3 INSTANTIATE LIBRARIES SECTION

By instantiating the libraries here we create global reference. You can note this by the commands that don't have *self*. in front of them. By convention we capitalize the names of global referenced libraries.

10.4.4.4 HANDLER CLASS section

The custom code is placed in a class so qtvcp can utilize it. This is the definitions on the handler class.

10.4.4.5 INITIALIZE section

Like all python libraries the *init* function is called when the library is first instantiated. You can set defaults and reference variables here. The widget references are not available at this point. The variables *halcomp*, *widgets* and *paths* give access to qtvcp's HAL component, widgets, and path info respectably. This is where you would set up global variables. Widgets are not actually accessible at this point.

10.4.4.6 SPECIAL FUNCTIONS section

There are several special functions that qtvcp looks for in the handler file. If qtvcp finds these it will call them, if not it will silently ignore them.

This function is called after the widgets and HAL pins are built You can manipulate the widgets and HAL pins or add more HAL pins here. Typically preferences can be checked and set, styles applied to widgets or status of linuxcnc be connected to functions. This is also where keybindings would be added.

Class patching allow you to override function calls in an imported module. Class patching must be done before the module is instantiated and it modifies all instances made after that. An example might be patching button calls from the G-code editor to call functions in the handler file instead. Class patching is also known as monkey patching.

This function is called to facilitate keyboard jogging etc. By using the keybindings library this can be used to easily add functions bound to keypresses.

This function gives raw key press events. It takes precedence over the `processed_key_event`.

This function gives raw key release events. It takes precedence over the `processed_key_event`.

This function is called just before the Qt event loop is entered. At the point all widgets/libraries/initialization code has completed and the screen is already displayed.

If present, this function overrides the normal function called when a user selects a total system shutdown. It could be used to do pre-shutdown housekeeping. The system will not shutdown if using this function, you will have to do that yourself. `qtvcp/linuxcnc` will shutdown without a prompt after this function returns

This function is called just before the screen closes. It can be used to do cleanup before closing.

10.4.4.7 STATUS CALLBACKS section

By convention this is where you would put functions that are callbacks from STATUS definitions.

10.4.4.8 CALLBACKS FROM FORM section

By convention this is where you would put functions that are callbacks from the widgets that you have connected to the MainWindow with the designer editor.

10.4.4.9 GENERAL FUNCTIONS section

By convention this is where you put your general functions

10.4.4.10 KEY BINDING section

If you are using the keybinding library this is where you place your custom key call routines. The function signature is:

```
def on_keycall_KEY(self,event,state,shift,cntrl):
    if state:
        self.do_something_function()
```

KEY being the code (from the keybindings library) for the desired key.

10.4.4.11 CLOSING EVENT section

Putting the close event function here will catch closing events. This replaces any predefined `closeEvent` function from `qtvcp` It's usually better to use the special `closing_cleanup__` function.

```
def closeEvent(self, event):
    self.do_something()
    event.accept()
```

10.4.5 Connecting widgets to python code

It's possible to connect widgets to python code using signals and slots. In this way you can give new functions to linuxcnc widgets or utilize standard widgets to control linuxcnc.

10.4.5.1 Overview

In the Designer editor you would create user function slots and connect them to widgets using signals. In the handler file you would create the slot's functions defined in Designer.

10.4.5.2 Using Designer to add slots

When you have loaded your screen into designer add a plain PushButton to the screen. You could change the name of the button to something interesting like *test_button*. There are two ways to edit connections - This is the graphical way. There is a button in the top tool bar of designer for editing signals. After pushing it, if you click-and-hold on the button it will show a arrow (looks like a ground signal from electrical schematic). Slide this arrow to a part of the main window that does not have widgets on it. A *Configure Connections* dialog will pop up. The list on the left are the available signals from the widget. The list on the right is the available slots on the main window and you can add to it.

Pick the signal *clicked()* - this makes the slots side available. click *edit* on the slots list. A *Slots/Signals of MainWindow* dialog will pop up. On the slots list at the top there is a plus icon - click it. you can now edit a new slot name. Erase the default name *slot()* and change it to *test_button()* press the ok button. You'll be back to the *Configure Connections* dialog. now you can select your new slot in the slot list. then press ok and save the file.

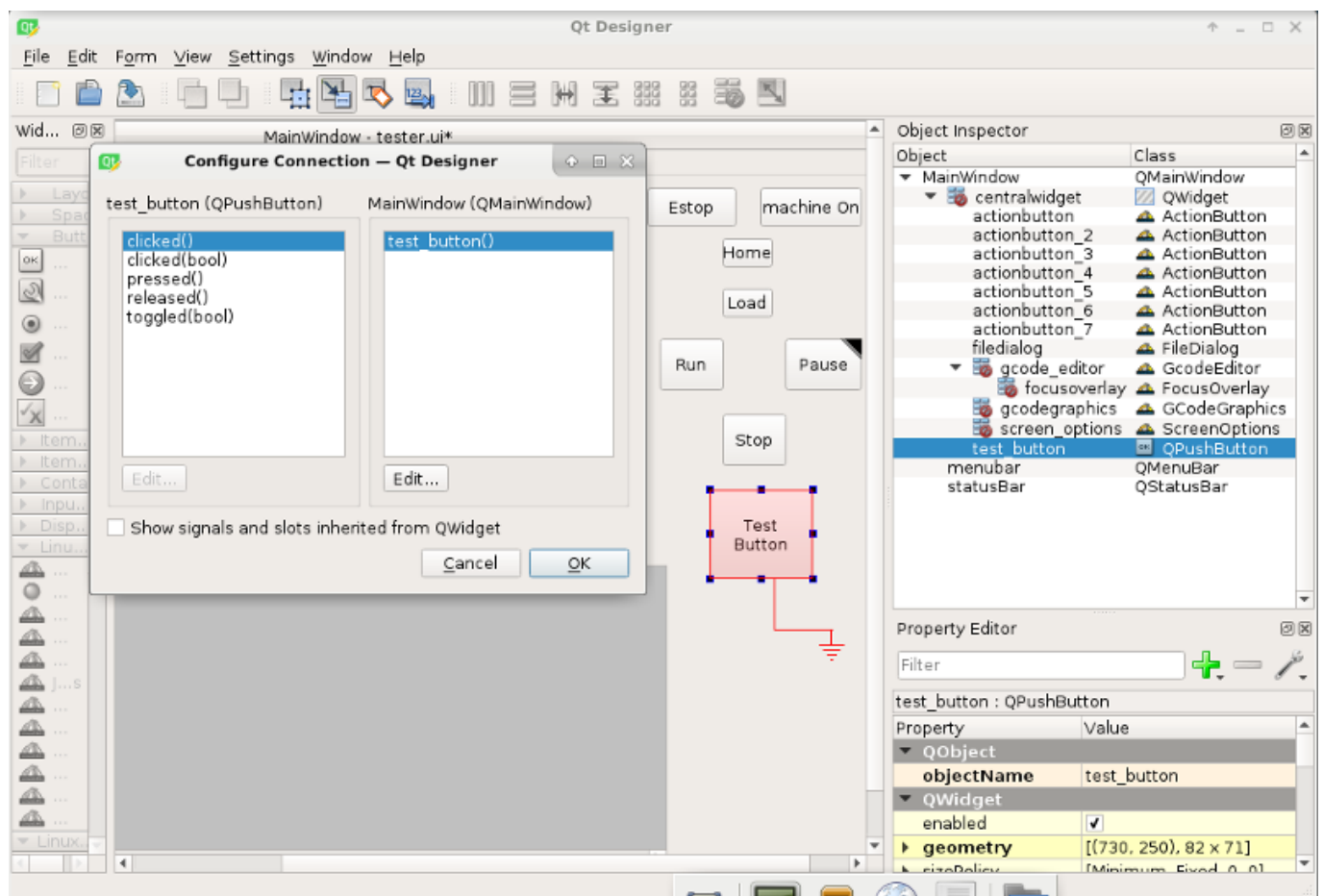


Figure 10.65: Designer signal/slot selection

10.4.5.3 Handler file changes

Now you must add the function to the handler file. The function signature is `def slotname(self)`: We will add some code to print the widget name.

So for our example:

```
def test_button(self):
    name = self.w.sender().text()
    print(name)
```

Add this code under the section named:

```
#####
# callbacks from form #
#####
```

In fact it doesn't matter where in the handler class you put the commands but by convention this is where to put it. Save the handler file. Now when you load your screen and press the button it should print the name of the button in the terminal.

10.4.5.4 More Information

[QtVCP Widgets](#)

[QtVCP Libraries](#)

[QtVCP Handler File Code Snippets](#)

[QtVCP Development](#)

[QtVCP Custom Designer Widgets](#)

10.4.6 QtVCP Virtual Control Panels

QtVCP is an infrastructure to display a custom CNC screen or control panel in LinuxCNC.

It displays a UI file built with the QTDesigner screen editor or combines this with python programming to create a VCP Panel.

QtVCP is completely customizable - you can add different buttons and status LEDs etc. or add python code for even finer grain customization.

10.4.6.1 Builtin panels

There are several builtin HAL panels available.
in a terminal type `qtvcp list` <return> to see a list.

used for copying QtVCP's builtin screen/panels/vismach code to a folder so one can customize it.

In a terminal:

```
qtvcp copy
```



Figure 10.66: copy Dialog - Copy QtVCP's builtin Screen, Panel or Vismach code

- This panel has a dial that adjusts S32 and Float HAL output pins.
- The dial's range can be adjusted from a drop down menu.
- The output can be scaled with the spinbox.
- A combobox can be used to automatically select and connect to a signal.

```
loadusr qtvcp test_dial
```

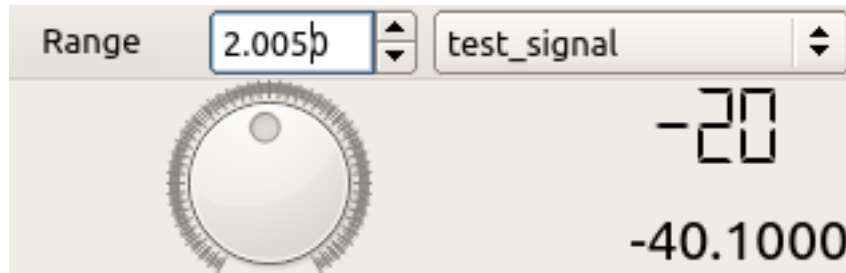


Figure 10.67: test dial - Test Dial VCP

- This panel has an LED that can be selected to watch HAL bit pins/signals.
- The LED's color can be adjusted from a drop down menu.
- The text box and state can be output as speech if sound is selected.
- A combobox can be used to automatically select and connect to a pin/signal.
- You can add more LEDs from the drop down menu.
- The LED can be detached from the main windows.

Here is how to load test_led from a HAL script.
 The -o switch sets how many LEDs the panel starts with.
 (If loading directly from a terminal omit the *loadusr*)

```
loadusr qtvcp test_led
loadusr qtvcp -o 4 test_led
```

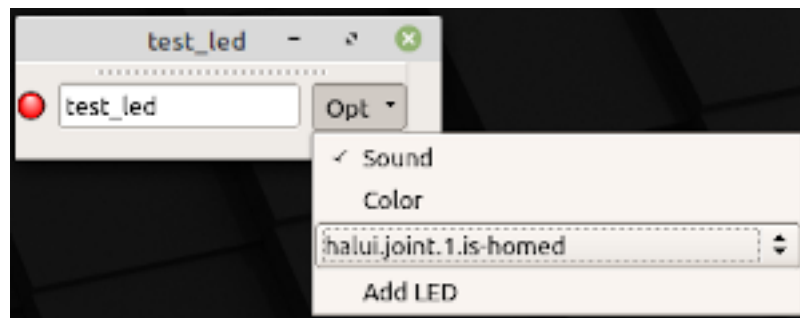


Figure 10.68: test dial - Test Dial VCP

- collect of useful widgets for testing HAL component. Including speech of LED state.

```
loadusr qtvcp test_panel
```



Figure 10.69: test panel - Test Panel VCP

- a camera display widget for rotational alignment



Figure 10.70: QtVCP cam align- Camera based alignment panel

- a small control panel to simulate MPG jogging controls etc. for simulated configurations

```
loadusr qtvcp sim_panel
```



Figure 10.71: QtVCP sim panel - Simulated controls for screen testing.

- 3d openGL view of a 3 axis milling machine

```
loadusr qtvcp vismach_mill_xyz
```

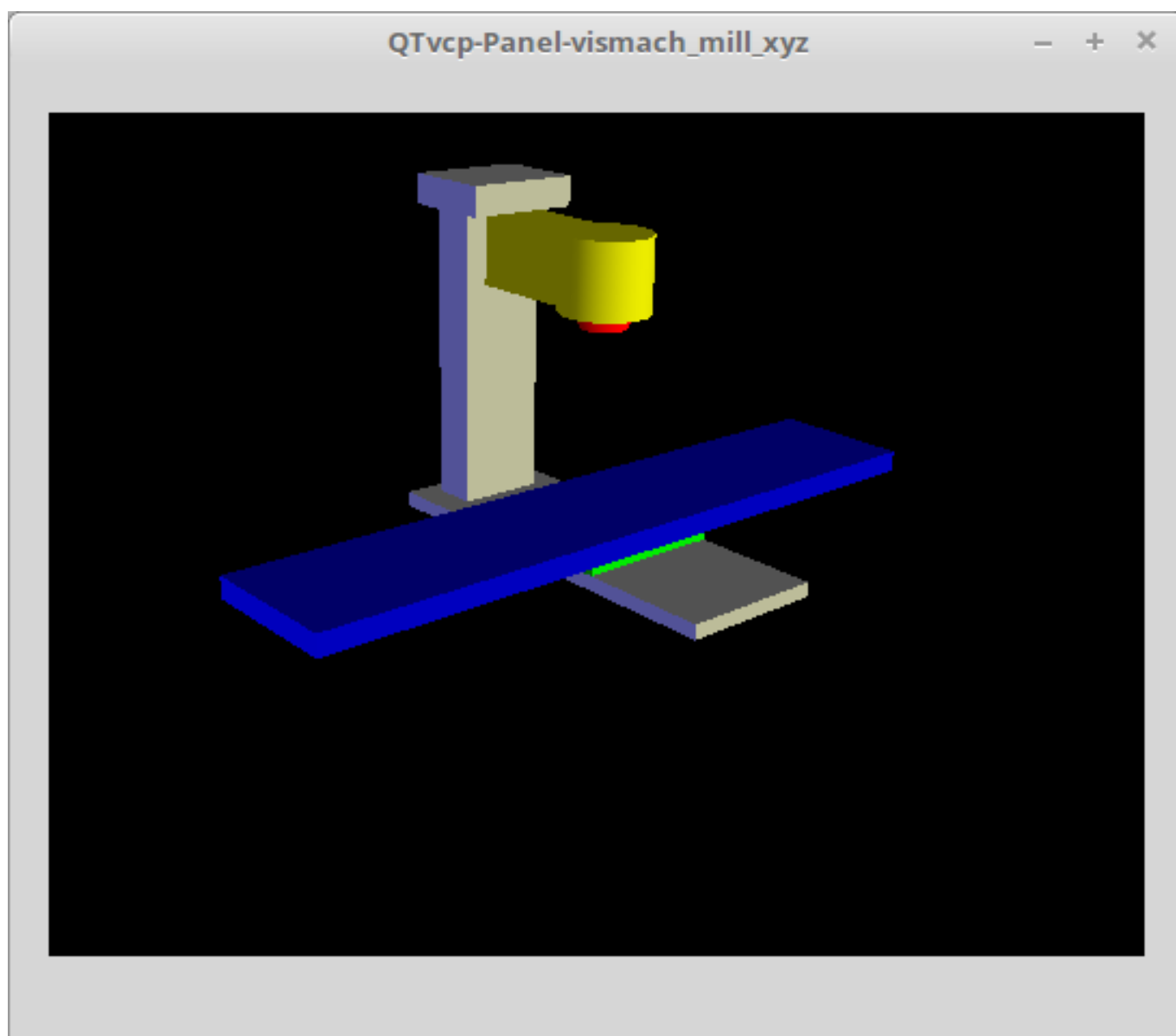



Figure 10.72: QtVCP vismach- 3 Axis Mill panel

- 3d openGL view of a scara based milling machine

```
loadusr qtvcp vismach_scara
```

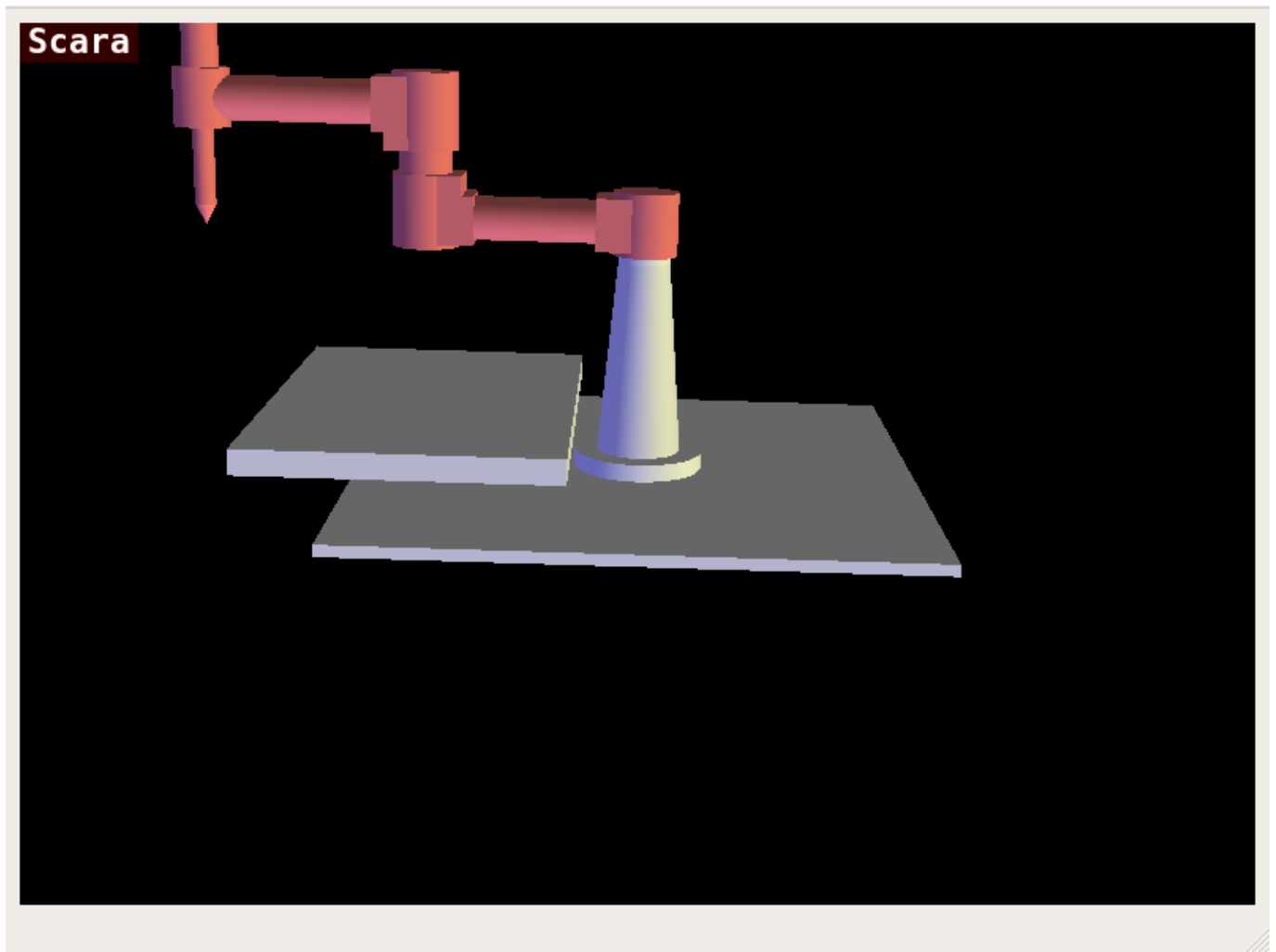


Figure 10.73: QtVCP vismach- 3 Axis Mill panel

- 3d openGL view of a 3 axis milling machine with an A axis/spindle

```
loadusr qtvcp vismach_millturn
```

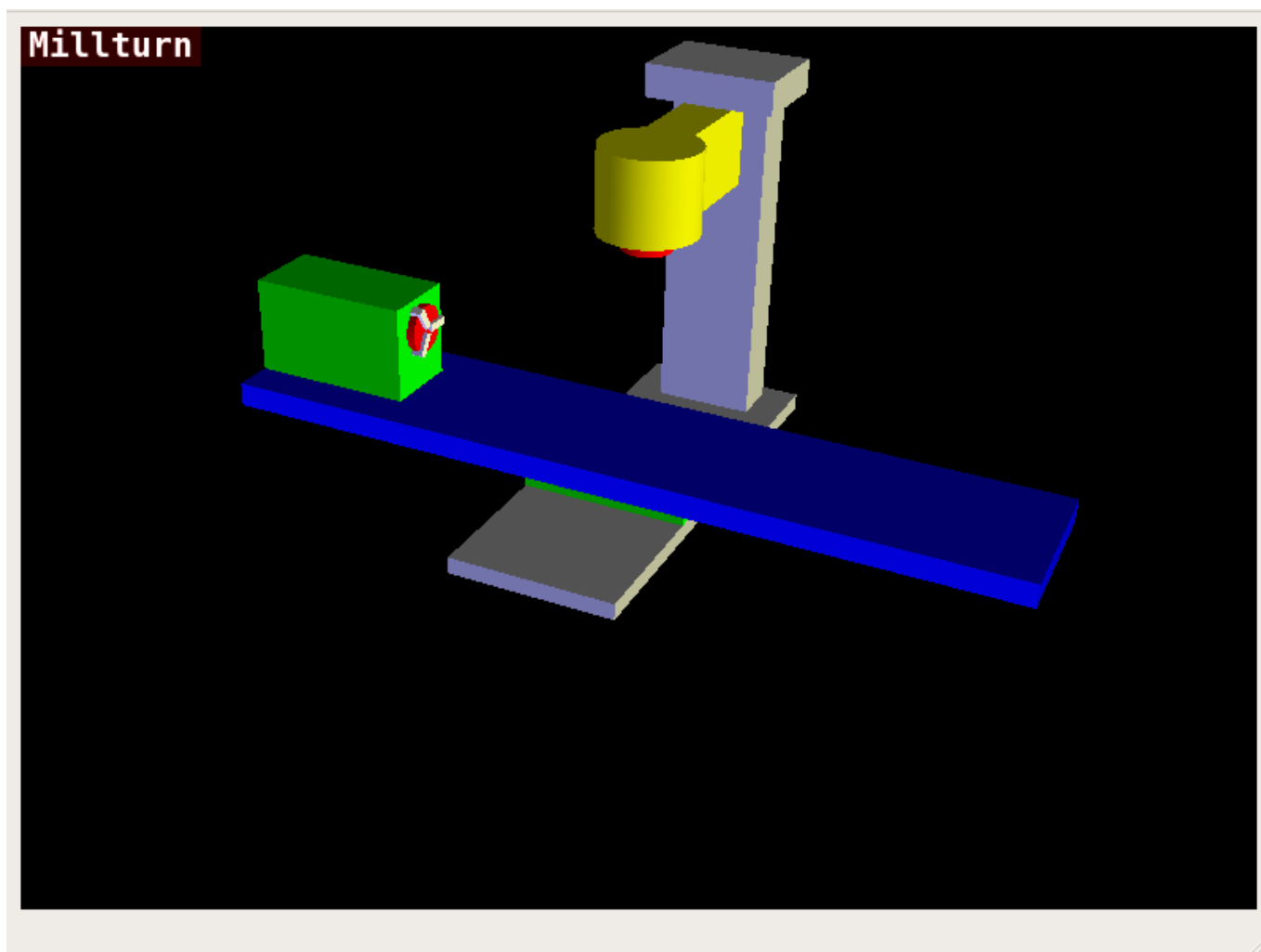


Figure 10.74: QtVCP vismach- 3 Axis Mill panel

- 3d openGL view of a 5 axis gantry type milling machine

```
loadusr qtvcp vismach_mill_5axis_gantry
```

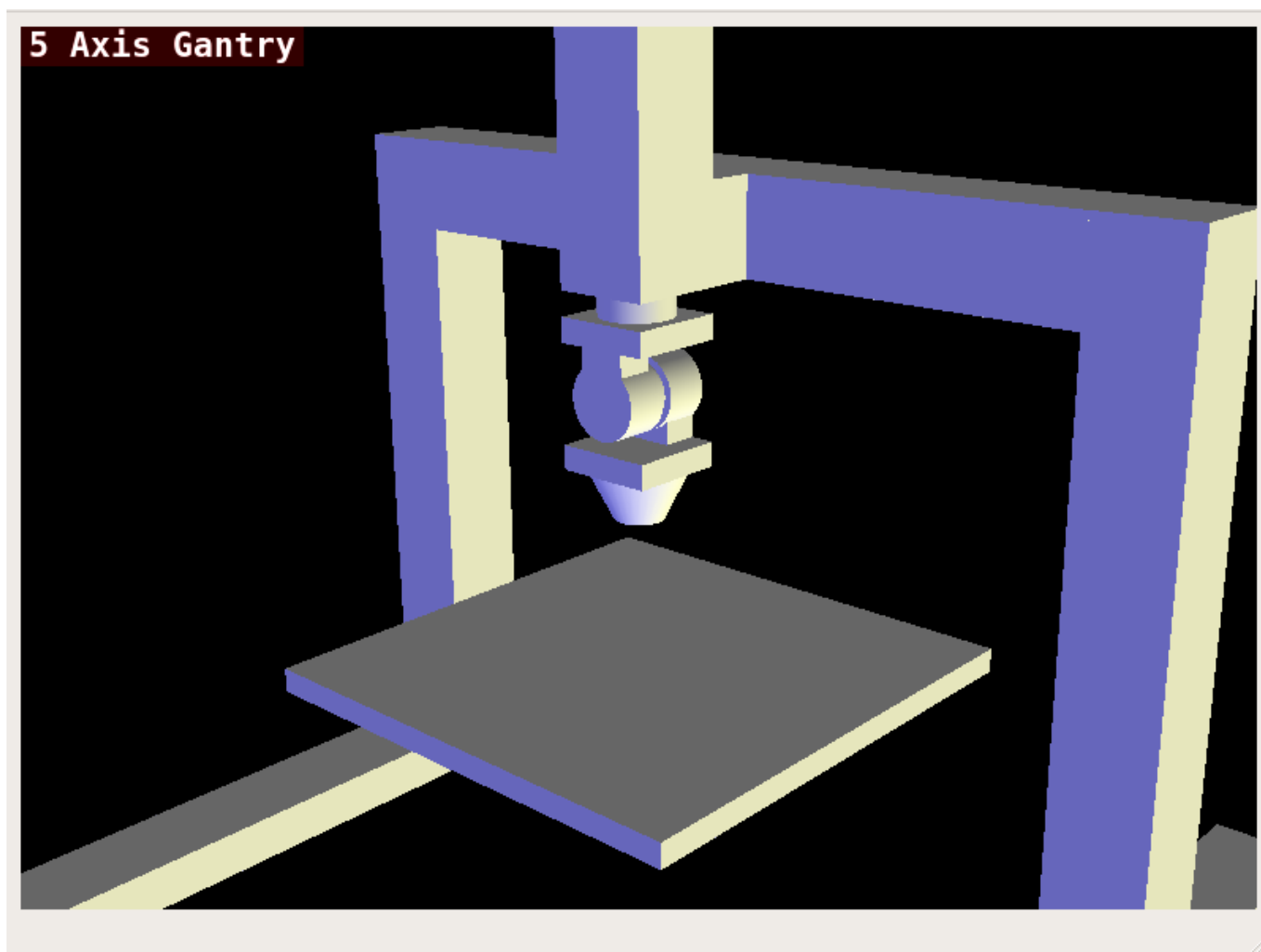


Figure 10.75: QtVCP vismach- 3 Axis Mill panel

- 3d openGL view of a 6 joint robotic arm.

```
loadusr qtvcp vismach_fanuc_200f
```

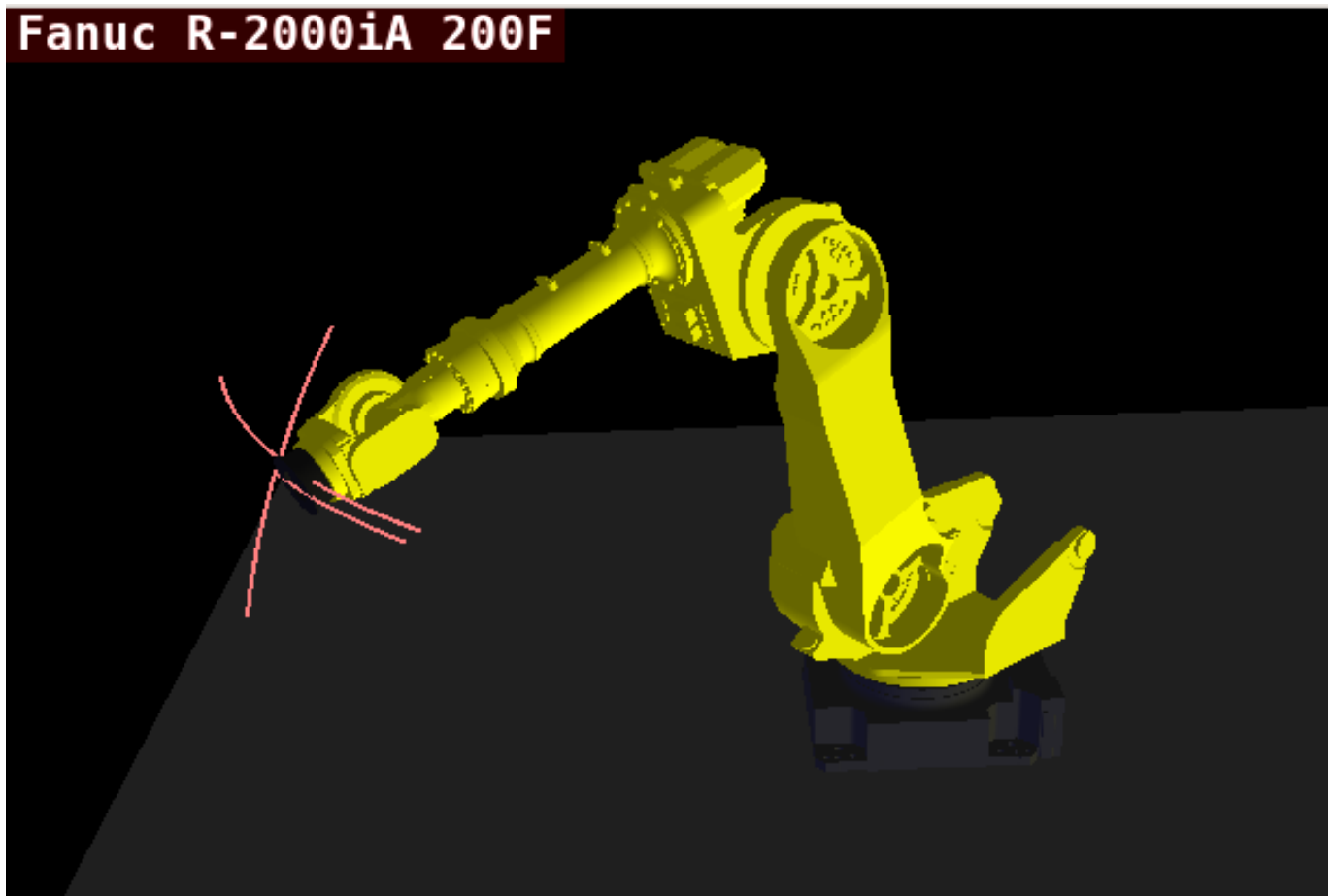


Figure 10.76: QtVCP vismach- 6 Joint Robotic Arm

- manual toolchange dialog that gives tool description.

```
loadusr -Wn tool_dialog qtvcp -o speak_on -o audio_on tool_dialog
```

Options:

- -o notify_on -use desktop notify dialogs instead of qtvcp native ones.
- -o audio_on -play sound on tool change
- -o speak_on -speak announcement of tool change

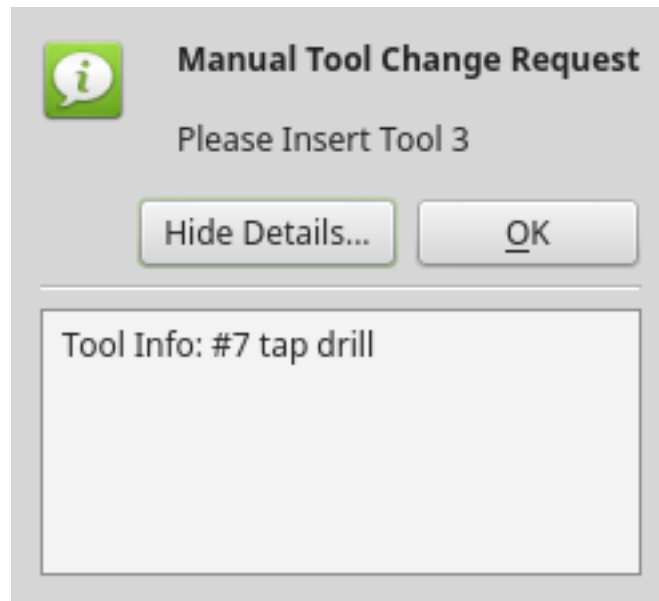


Figure 10.77: QtVCP tool dialog - Manual Tool Change Dialog

10.4.6.2 Custom Panels

You can of course make your own panel and load it.

If you made a ui file named *my_panel.ui* and name the following HAL file, *my_panel.hal*

You would then load this from a terminal with `halrun -I -f my_panel.hal`

```
# load realtime components
loadrt threads
loadrt classicladder_rt

# load user space programs
loadusr classicladder
loadusr -Wn my_panel qtvcp my_panel.ui

# add components to thread
addf classicladder.0.refresh thread1

# connect pins
net bit-input1      test_panel.checkbox_1      classicladder.0.in-00
net bit-hide        test_panel.checkbox_4      classicladder.0.hide_gui
net bit-output1     test_panel.led_1           classicladder.0.out-00
net s32-in1         test_panel.doublescale_1-s classicladder.0.s32in-00

# start thread
start
```

In this case we load qtvcp using `-Wn`; which waits for the panel to finish loading before continuing to run the next HAL command. This is so the HAL pins from the panel are finished in case they are used in the rest of the file.

10.4.6.3 More Information

[QtVCP Widgets](#)

[QtVCP Libraries](#)

[QtVCP Handler File Code Snippets](#)

[QtVCP Development](#)

[QtVCP Custom Designer Widgets](#)

10.4.7 QtVCP Widgets

QtScreen uses QtVCP widgets for LinuxCNC integration. Widget is the general name for the UI objects such as buttons and labels in QTPy. You are free to use any available widgets in the QTDesigner editor. There are also special widgets made for LinuxCNC that make integration easier. These are split in two headings on the right side of the editor. One is for HAL only widgets. The other is for CNC control widgets. You are free to mix them in any way on your panel.

Note

This description of widget properties can easily be out of date due to further development and lack of people to write docs (a good way to give back to the project). The definitive descriptions are found by looking in the source code.

10.4.7.1 HAL Only Widgets

These Widgets usually have HAL pins and don't react to the machine controller.

Allows one to embed program into the widget. Only programs that utilize the xembed protocol will work such as:

- GladeVCP virtual control panels
- Onboard virtual keyboard
- qtvcp virtual control panels
- mplayer video player

Allows one to adjust a HAL pins using a sliding pointer.

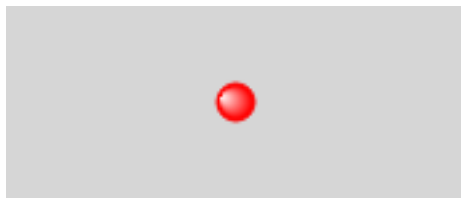


Figure 10.78: LED

An indicator that optionally follows a HAL pin's logic.

- `halpin_option` -selects if the LED follows an input HAL pin or program state.
-

- diameter -diameter of the LED
- color - Color of the LED when on.
- off_color - Color of the LED when off.
- alignment - Qt Alignment hint.
- state -current state of LED
- flashing -turns flashing option on and off.
- flashRate -sets the flash rate.

The LED properties can be defined in a stylesheet with the following code added to the .qss file. The name_of_led would be the name defined Designer's editor.

```
LED #name_of_led{
    qproperty-color: red;
    qproperty-diameter: 20;
    qproperty-flashRate: 150;
}
```

This widget allows the user to check a box to set a HAL pin true or false.

It is based on PyQt's QCheckBox

This widget allows a user to set HAL pins true or false. Only one widget of a group can be true at a time.

It is based on PyQt's QRadioButton.

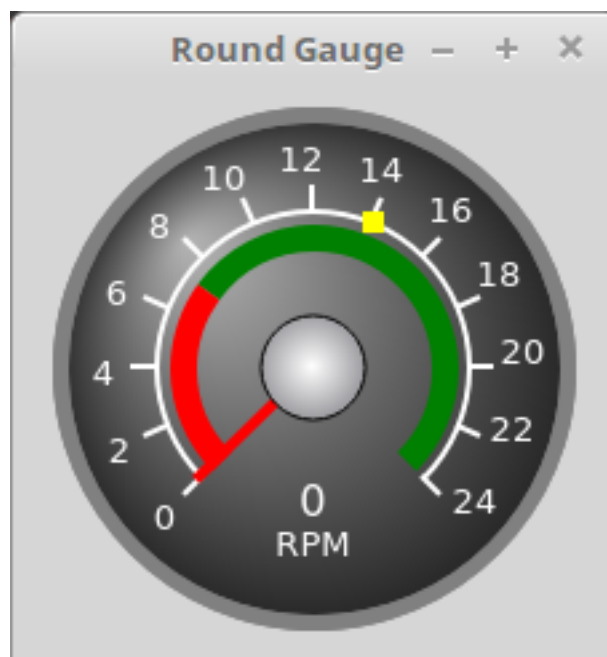


Figure 10.79: Round Gauge

Round Gauge can be used in a LinuxCNC GUI to display an input parameter on the dial face. There are several properties that are user settable in order to customize the appearance of the gauge.

There are 2 inputs that are not customizable. They can be set via HAL pins, programmatically or via signals from other widgets. The non customizable parameters are:

- *Value* - This is the input value that will be displayed with the gauge needle and in the digital readout. It must be set to a value between 0 and the maximum value.
- *Setpoint* - This is a value that determines the location of a small marker on the gauge face. It must be set to a value between 0 and the maximum value.

The following parameters can be set either programmatically or via the designer property editor. The custom parameters are:

- *halpin_option* - Setting this True will create 2 HAL pins. One is for setting the value input and the other is for setting the setpoint. If this option is not set, then value and setpoint must be connected programmatically, ie., in the handler file.
 - *max_reading* - This value determines the highest number that will be displayed on the gauge face.
 - *max_value* - This is the maximum expected value of the value input signal. In other words, it is the full scale input.
 - *num_ticks* - This is the number of ticks, or gauge readings that will be displayed on the gauge face. It should be set to a number that ensures the text readings around the gauge face are readable. The minimum allowed value is 2.
 - *zone1_color* - Zone1 extends from the maximum reading to the threshold point. It can be set to any RGB color.
 - *zone2_color* - Zone2 extends from the threshold point to the minimum reading, which is 0. It can be set to any RGB color.
 - *bezel_color* - This is the color of the outer ring of the gauge.
 - *threshold* - The threshold is the transition point between the zones. It should be set to a value between 0 and the maximum value. The maximum allowed value is set to the gauge maximum value and minimum value is 0.
 - *gauge_label* - This is the text that appears below the value readout, near the bottom of the gauge. The function of the gauge is then easily visible.
-

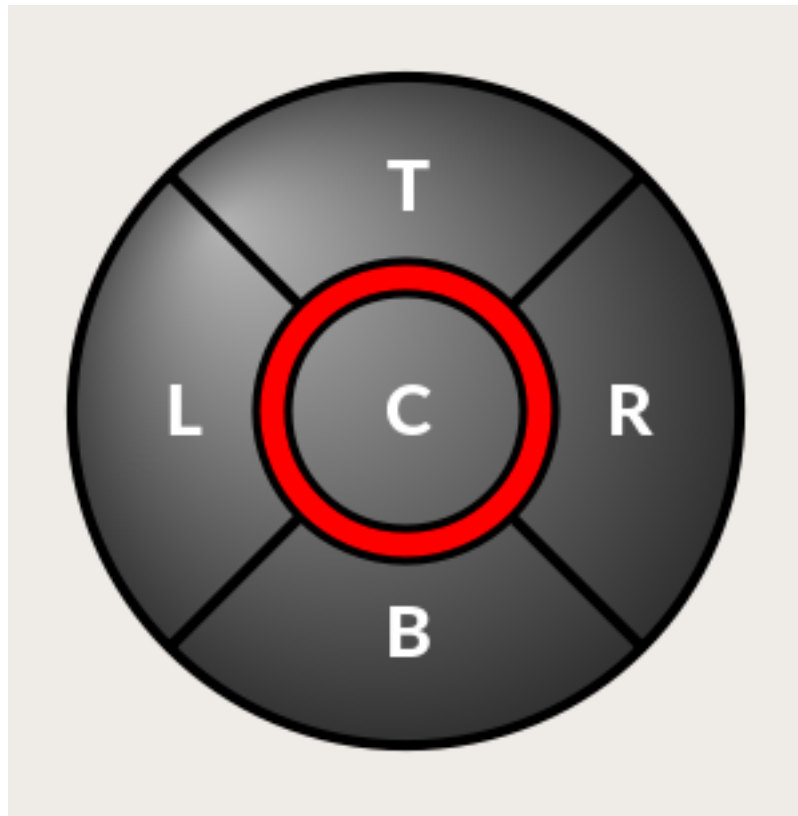


Figure 10.80: HALPAD

This widget looks and acts like a 5 button D-pad, with an LED ring. Each button has an selectable type (Bit, S32 or Float) output HAL pin. The LED center ring has selectable colors for off and on and is controlled by a Bit HAL pin.

There are enumerated constants used to reference indicator positions.

```
NONE
LEFT
RIGHT
CENTER
TOP
BOTTOM
LEFTRIGHT
TOPBOTTOM
```

There are constants for HAL pin type:

```
NONE
BIT
S32
FLOAT
```

You use the widget Designer name plus the reference constant.

```
self.w.halpadname.set_highlight(self.w.halpadname.LEFTRIGHT)
```

- *pin_name*: Optional name to use for the HAL pins basename. If left blank, the designer widget name will be used.

- *pin_type*: Select the HAL output pin type. This property is only used at startup. Selection can be set in Designer:

```
NONE
BIT
S32
FLOAT
```

- *left_image_path*:
- *right_image_path*:
- *center_image_path*:
- *top_image_path*:
- *bottom_image_path*:
A file path or resource path to an image to display in the described button location. If the reset button is pressed in the Designer editor property, the image will not be displayed. (allowing optionally text)
- *left_text*:
- *right_text*:
- *center_text*:
- *top_text*:
- *bottom_text*:
A text string to be displayed in the described button location. If left blank an image can be designated to be displayed.
- *true_color*:
- *false_color*:
Color selection for the center LED ring to be displayed when the *BASENAME.light.center* HAL pin is True or False.
- *text_color*:
Color selection for the button text.
- *text_font*:
Font selection for the button text.

The above properties could be set in styles sheets.

```
HALPad{
  qproperty-on_color: #000;
  qproperty-off_color: #444;
}
```

This widget allows a user to set a HAL pin true or false. as an option it can be a toggle button. It also has other options:

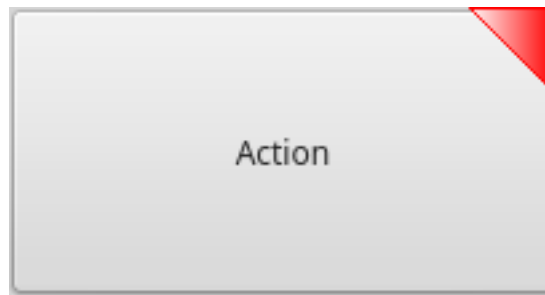


Figure 10.81: Indicated Action Button

Indicator_option puts a *LED* on the top of the button. It can be a triangle, circle, top bar or side bar. The size and position can be adjusted. It will indicate the current state of the button, the state of a HAL pin or LinuxCNC status. Use properties to customize the indicator (not all are applicable to every LED shape).

```
on_color
off_color
indicator_size
circle_diameter
shape_option
right_edge_offset
top_edge_offset
height_fraction
width_fraction
corner_radius
```

The LED indicator color can be defined in a stylesheet with the following code added to the .qss file.

```
Indicated_PushButton{
    qproperty-on_color: #000;
    qproperty-off_color: #444;
}
```

or for a particular button:

```
Indicated_PushButton #button_estop{
    qproperty-on_color: black;
    qproperty-off_color: yellow;
}
```

Indicated PushButtons have exclusive options:

- indicator_HAL_pin_option
- indicator_status_option

Indicator_HAL_pin_option will add a halpin, using the button name *-led*, that controls the button indicator state.

indicator_status_option will make the LED indicate the state of these selectable LinuxCNC status:

```
Is Estopped
Is On
All Homed
Is Joint Homed
Idle
Paused
```

```

Flood
Mist
Block Delete
Optional Stop
Manual
MDI
Auto
Spindle Stopped
Spindle Fwd
Spindle Reverse
On Limits

```

The `some_indicator_status_options` holds a property that can be used with a stylesheet to change the color of the button based on the state of the property in LinuxCNC. Currently these status options can be used to auto style buttons: `is_estopped_status` indicated buttons change the property `isEstopped` `is_on_status` indicated buttons change the property `isStateOn` `manual,mdi,auto_status` indicated buttons change the properties `isManual`, `isMDI`, `isAuto`.

Here is a sample stylesheet entry. It sets the background of mode button widgets when LinuxCNC is in that mode.

```

ActionButton[isManual=true] {
    background: red;
}
ActionButton[isMdi=true] {
    background: blue;
}
ActionButton[isAuto=true] {
    background: green;
}

```

Here is how you specify a particular widget - by it's `objectName` in designer.

```

ActionButton #estop button [isEstopped=false] {
    color: yellow;
}

```

Choosing the `checked_state_text_option` allows a *checkable* button to change the text based on it's checked state. It uses the properties `true_state_string` and `false_state_string` to specify the text for each state.

`\\n` will be converted to a newline.

You can set/change these in style sheets:

```

ActionButton #action_aux{
    qproperty-true_state_string: "Air\\nOn";
    qproperty-false_state_string: "Air\\nOff";
}

```

The `python_command_option` allow small snippets of python code to be run from the push of a button, with out having to edit the handler file. (though it can call functions in the handler file) When using the `command_string` properties.

- `true_python_cmd_string` - a python command that will be called when the button is toggled true
- `false_python_cmd_string` - a python command that will be called when the button is toggled false

The capitalized word *INSTANCE* will give access to the widgets instances and handler functions. eg. `INSTANCE.my_handler_function_call(True)`.

The capitalized word *ACTION* will give access to qtvcap's ACTION library. eg. `ACTION.TOGGLE_FLOOD()`.

The capitalized word `PROGRAM_LOADER` will give access to qtvcp's `PROGRAM_LOADER` library. eg. `PROGRAM_LOADER.load_halshow()`.

The capitalized word `HAL` will give access to HAL's python module. eg. `HAL.set_p('motion.probe-input,1)'`.

It is based on PyQt's `QPushButton`.

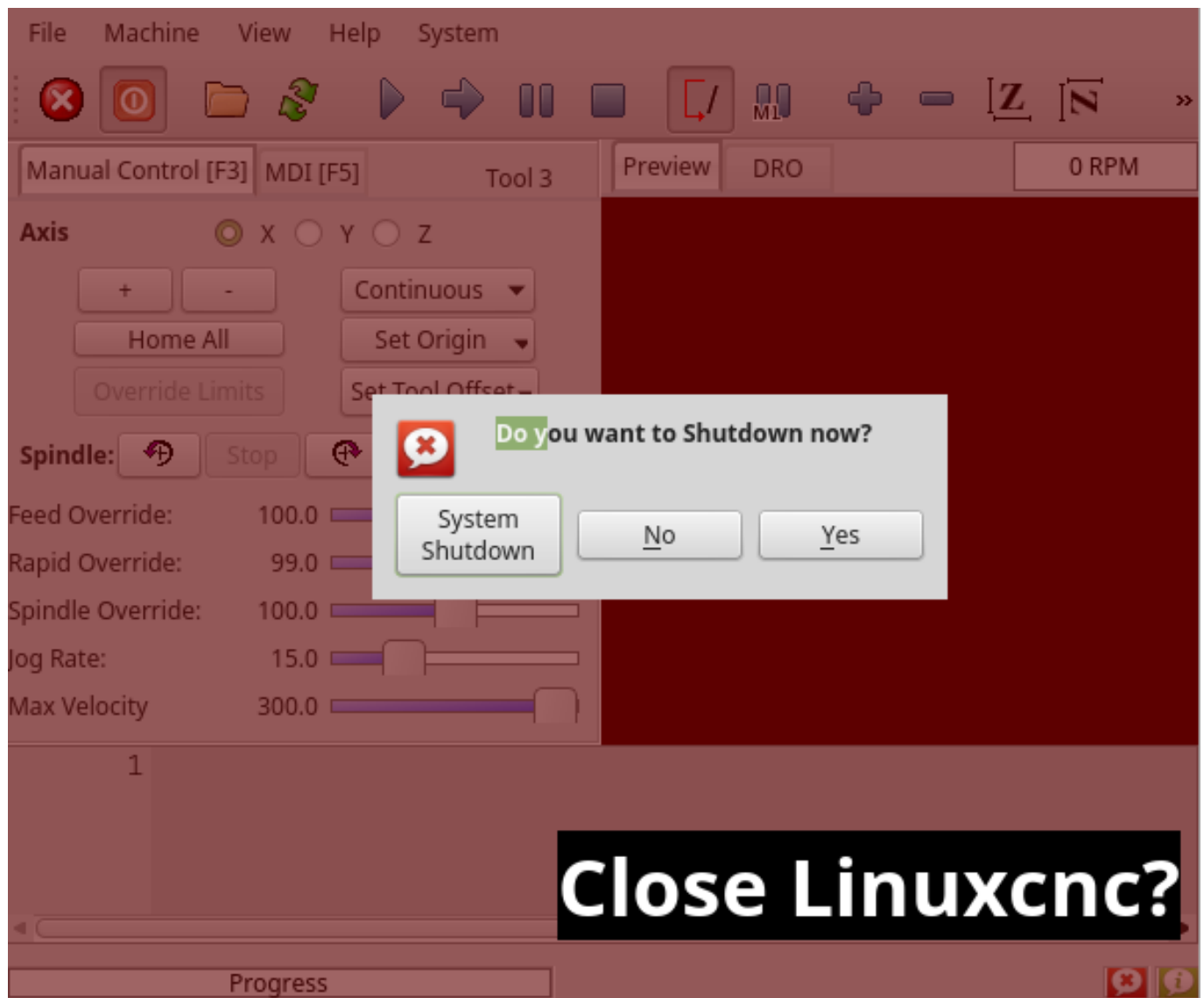


Figure 10.82: Focus overlay example for confirm close prompt

This widget places a coloured overlay over the screen usually while a dialog is showing. Used to create a *focused* feel and to draw attention to critical information. It can also show a translucent image. It can also display message text and buttons. This widget can be controller with STATUS messages.

This widget controls if the widgets inside it are enabled or disabled. disabled widgets are typically a different colour and do not respond to actions.

It is based on PyQt's `QGridLayout`.

This widget displays values sent to it from HAL pins, programically or a `QtSignal`. The input pin can be selected as Bit, S32, Float or no pin selected. There is a text Template property to set the rich text

and/or to format the text. Basic formatting might be, for bool: %r, for integer: %d, for float: %0.4f. A rich text example might be:

```
self.w.my_hal_label.setProperty(textTemplate, """
<html><head/><body><p><span style=" font-size:12pt;
font-weight:600; color:#f40c11;">%0.4f</span></p></body></html>
""")
```

The *setDisplay* slot can be connected to a integer, float or bool signal. If the property *pin_name* is not set the widget name will be used.

There are function calls to display values:

- [HALLabelName].setDisplay(some_value) can be used to set the display if no HAL pin is selected.
- [HALLabelName].setProperty(textTemplate,"%d") - set the template of the display.

It is based on PyQt's QLabel

This widget displays HAL float/s32/bit values in a LCD looking way. It can display numbers in decimal, hexadecimal, binary and octal formats by setting the property *mode*. When using floats you can set a formatting string. You must set the property *digitCount* to an appropriate setting to display the largest number.

- *pin_name*: Option string to be used as the HAL pin name. If set to an empty string the widget name will be used.
- *bit_pin_type*: Selects the input pin as type BIT.
- *s32_pin_type*: Selects the input pin as type S32.
- *float_pin_type*: Selects the input pin as type FLOAT.
- *floatTemplate*: A string that will be used as a python3 format template to tailor the LCD display. Only used when a FLOAT pin is selected. eg `{:.2f}` will display a float rounded to 2 numbers after the decimal. A blank setting will allow the decimal to move as required.

It is based on PyQt's QLCDNumber.

This widget is a spin button entry widget. used for setting a s32 and float HAL pin. It has an internal scale factor, set to a default of 1, that can be set programmatically or using a QtSignal. The scale defaults to 1 the *setInput* slot can be connected to a integer, or float signal.

There is a function call to change the internal scaling factor:

- [HALLabelName].setInput(some_value)

The HAL pins will be set to the value of the internal scale times the widget displayed value.

This widget displays a image from a web camera. It overlays an adjustable circular and cross hair target over the image. Camview was built with precise visual positioning in mind.

This widget is used to connect an arbitrary Qt widget to HAL using signals/slots. It is used for widgets that should respond to HAL pin changes.

This widget is used to connect an arbitrary Qt widget to HAL using signals/slots. It is used for widgets that should control HAL pins.

This is used to switch the view of a multi-widget layout to show just one widget. This might be used to flip between a large view of a widget or a smaller multi widget view. It's different from a stacked widget as it can pull a widget from anywhere in the screen and place it in it's page with a different layout then it originally had. The original widget must be in a layout for switcher to put it back.

In Designer you will add the widgetswitcher widget on screen. Right click the widgetswitcher and add a page, then populate it with widgets/layouts you wish to see in a default form. Then add as many pages as there are views to switch to. on each page add a layout widget. After adding the layout you must right click the widget switcher again and set the layout option. click on the widgetswitcher widget and then scroll to the bottom of the property editor. you are looking for the dynamic property *widget_list*. double click the to the right of the *widget_list* property. A dialog will pop up allowing you to add the names of the widgets to move to the pages you added to the widgetswitcher.

There are function calls to display specific widgets:

- [WidgetSwitcherName].show_id_widget(number)
- [WidgetSwitcherName].show_named_widget(widget_name)
- [WidgetSwitcherName].show_default()
- [WidgetSwitcherName].show_next()

By calling one of these functions, you control what widget is currently displayed. *show_default()* shows the page 0 layout, and puts all other widgets back to where they were as initially built in Designer.

It is based on the QStack widget.

10.4.7.2 Machine Controller Widgets

These widgets interact to the Machine Controller state.

These buttons are used to control action of the machine controller. They are built on top of indicator_buttons so can have LEDs overlaid.

Note

If you left double click on this widget you can launch a dialog to set any of these action. The dialogs will help to set the right related data to the selected action. You can also change these properties directly in the property editor.

You can select one of these actions:

- *Estop*
 - *Machine On*
 - *Auto*
 - *mdi*
 - *manual*
 - *run*
 - *run_from_line status* (gets line number from STATUS message gcode-line-selected)
 - *run_from_line slot* (gets line number from designer int/str slot setRunFromLine)
 - *abort*
 - *pause*
 - *load dialog* (requires a dialog widget present)
 - *Camview dialog* (requires camview dialog widget present)
-

- *origin offset dialog* (requires origin offset dialog widget present)
 - *macro dialog* (requires macro dialog widget present)
 - *Launch Halmeter*
 - *Launch Status*
 - *Launch Halshow*
 - *Home* (set the joint number to -1 for all-home)
 - *Unhome* (set the joint number to -1 for all-unhome)
 - *Home Selected* Homes the joint/axis selected by STATUS
 - *Unhome Selected* Unhomes the joint/axis selected by STATUS
 - *zero axis*
 - *zero G5X* zeros the current user coordinate system offsets
 - *zero G92* zeros the optional G92 offsets
 - *zero Z rotational* zeros the rotation offset
 - *jog joint positive* (set the joint number)
 - *jog joint negative* (set the joint number)
 - *jog selected positive* (selected with a different widget or STATUS)
 - *jog selected negative* (selected with a different widget or STATUS)
 - *jog increment* (set metric/imperial/angular numbers)
 - *jog rate* (set the float/alt float number)
 - *feed override* (set the float/alt float number)
 - *rapid override* (set the float/alt float number)
 - *spindle override* (set the float/alt float number)
 - *spindle fwd*
 - *spindle backward*
 - *spindle stop*
 - *spindle up*
 - *spindle down*
 - *view change* (set view_type_string)
 - *limits override*
 - *flood*
 - *mist*
 - *block delete*
 - *optional stop*
 - *mdi command* - calls a hard code MDI command (set command_string)
 - *INI mdi number* calls an INI based MDI command (set ini_mdi_number)
-

- *dro absolute*
- *dro relative*
- *dro dtg*
- *exit screen* Closes down LinuxCNC
- *Override limits* Temporarily override hard limits
- *launch dialogs* pops up dialogs if they are included in ui file.
- *set DRO to relative*
- *set DRO to absolute*
- *set DRO to distance-to-go*

These set attributes of the selected action. Availability depends on the widget.

- *toggle float option* - allows jog rate and overrides to toggle between two rates
- *joint number* - selects the joint/axis that the button controls
- *incr imperial number* - sets the imperial jog increment (set negative to ignore)
- *incr mm number* -sets the metric jog increment (set negative to ignore)
- *incr angular number* -sets the angular jog increment (set negative to ignore)
- *float number* - used for jograte and overrides
- *float alternate number* -for jograte and overrides that can toggle between two float numbers
- *view type string* - can be p, x, y, y2, z, z2, clear, zoom-in, zoom-out, pan-up, pan-down, pan-left, pan-right, rotate-up, rotate-down, rotate-cw, rotate-ccw
- *command string* - MDI command string that will be invoked if the MDI command action is selected.
- *ini mdi number* - a reference to the INI file [MDI_COMMAND_LIST] section. Set an integer of select one line under the INI's MDI_COMMAND line starting at 0. Then in the INI file, under the heading [MDI_COMMAND_LIST] add appropriate lines. The commands are separated by the ; The label is set after the comma. The symbols \n adds a line break.

```
[MDI_COMMAND_LIST]
MDI_COMMAND = G0 Z25;X0 Y0;Z0, Goto\nUser\nZero
MDI_COMMAND = G53 G0 Z0;G53 G0 X0 Y0,Goto\nMachn\nZero
```

Action buttons are subclassed from indicated_PushButton

Indicator_option puts a *LED* on the top of the button. It can be a triangle, circle, top bar or side bar. The size and position can be adjusted It will indicated the current state of the button, the state of a HAL pin or LinuxCNC status. Use properties to customized the indicator (not all are applicable to every LED shape).

```
on_color
off_color
indicator_size
circle_diameter
shape_option
right_edge_offset
top_edge_offset
height_fraction
width_fraction
corner_radius
```

The LED indicator color can be defined in a stylesheet with the following code added to the .qss file.

```
Indicated_PushButton{
    qproperty-on_color: #000;
    qproperty-off_color: #444;
}
```

or for a particular button:

```
Indicated_PushButton #button_estop{
    qproperty-on_color: black;
    qproperty-off_color: yellow;
}
```

Indicated PushButtons have exclusive options:

- indicator_HAL_pin_option
- indicator_status_option

Indicator_HAL_pin_option will add a halpin, using the button name *-led*, that controls the button indicator state.

indicator_status_option will make the LED indicate the state of these selectable LinuxCNC status:

```
Is Estopped
Is On
All Homed
Is Joint Homed
Idle
Paused
Flood
Mist
Block Delete
Optional Stop
Manual
MDI
Auto
Spindle Stopped
Spindle Fwd
Spindle Reverse
On Limits
```

Choosing the checked_state_text_option allows a *checkable* button to change the text based on it's checked state. It uses the properties *true_state_string* and *false_state_string* to specify the text for each state.

\\n will be converted to a newline.

You can set/change these in style sheets:

```
Indicated_PushButton #auxiliary {
    qproperty-true_state_string: "Air\\nOn";
    qproperty-false_state_string: "Air\\nOff";
}
```

The python_command_option allow small snippets of python code to be run from the push of a button, with out having to edit the handler file. (though it can call functions in the handler file) When using the command_string properties.

- *true_python_cmd_string* - a python command that will be called when the button is toggled true

- *false_python_cmd_string* - a python command that will be called when the button is toggled false

The capitalized word *INSTANCE* will give access to the widgets instances and handler functions. eg. *INSTANCE.my_handler_function_call(True)*

The capitalized word *ACTION* will give access to qtvcp's ACTION library. eg. *ACTION.TOGGLE_FLOOD()*

The capitalized word *PROGRAM_LOADER* will give access to qtvcp's PROGRAM_LOADER library. eg. *PROGRAM_LOADER.load_halshow()*

The capitalized word *HAL* will give access to HAL's python module. eg. *HAL.set_p('motion.probe-input,1)'+* Indicated PushButtons and Actionbuttons are based on PyQt's QPushButton

Action tool buttons are similar in concept to action buttons, but they use QToolButtons to allow optional actions to be selected by pushing and holding the button till the option menu pops up.

Currently there is only one option - user view.

It is based on PyQt's QToolButton.

User view tool button allows a user to record and return to a arbitrary graphics view. Press and hold the button to have the menu pop up and press *record view*. This records the currently displayed graphics view. click the button normally to return to the last recorded position.

The position recorded position will be remembered at shutdown if a preference file option is set up.

Note

Do to programming limitations, the recorded position may not show exactly the same, Particularly if you pan zoomed out and pan again zoomed in while setting the desired view. Best practice is to select a main view, modify as desired, record, then immediately click the button to *return* to the recorded position. If it is not as you like, modify it's existing position and re-record.

Round buttons work the same as ActionButtons other then the button is cropped round. They are intended only to be visually different. They have two path properties for displaying images on true and false.

This allows one to select and set an AXIS. If the button is set checkable, it will indicate which axis is selected. If you press and hold the button a pop up menu will show allowing one to:

- Zero the axis
- divide the axis by 2
- set the axis arbitrarily
- reset the axis to the last number recorded

You select the axis by setting the joint number You can select a halpin option that is set true when the axis is selected

It is based on PyQt's QToolButton

This is used to align the work piece or zero part features using a webcam. It uses opencv vision library. This will display the current position of an axis.

- *Qjoint_number* - joint number of offset to display (10 will specify rotational offset)
 - *Qreference_type* - actual, relative or distance to go (0,1,2)
 - *metric_template* - format of display ie *%10.3f*
 - *imperial_template* - format of display ie *%9.4f*
 - *angular_template* - format of display ie *%Rotational: 10.1f*
-

The DRO_Label widget holds a property *isHomed* that can be used with a stylesheet to change the color of the DRO_Label based on home state of the joint number in LinuxCNC.

Here is a sample stylesheet entry. It sets the font of all DRO_Label widgets. It sets the text template (to set resolution) of the DRO Then sets the text color based on the Qt *isHomed* Property.

```
DROLabel {
    font: 25pt "Lato Heavy";
    qproperty-imperial_template: '%9.4f';
    qproperty-metric_template: '%10.3f';
    qproperty-angular_template: '%11.2f';
}

DROLabel[isHomed=false] {
    color: red;
}

DROLabel[isHomed=true] {
    color: green;
}
```

Here is how you specify a particular widget - by it's objectName in designer.

```
DROLabel #dr0_x_axis [isHomed=false] {
    color: yellow;
}
```

It is based on PyQt's QLabel

This displays G-code in text form. It will highlight the currently running line. This can also display MDI history when LinuxCNC is in MDI mode. This can also display log entries when LinuxCNC is in MANUAL mode. This will also display preference file entries if you enter *PREFERENCE* in capitals into the MDIline widget. It has a signal percentDone(int) that that can be connected to a slot (such as a progressBar to display percent run)

- `auto_show_mdi_status`
Set true to have the widget switch to MDI history when in MDI mode
- `auto_show_manual_status`
Set true to have the widget switch to machine log when in Manual mode

The GcodeDisplay properties can be set in a stylesheet with the following code added to the .qss file.

```
EditorBase{
    qproperty-styleColorBackground: lightblue;
    qproperty-styleColor0: black;
    qproperty-styleColor1: #000000; /* black */
    qproperty-styleColor2: red;
    qproperty-styleColor3: black;
    qproperty-styleColor4: yellow;
    qproperty-styleColorMarginText: White;
    qproperty-styleColorMarginBackground: blue;
    qproperty-styleFont0: "Times,12,-1,0,90,0,0,0,0,0";
    qproperty-styleFont1: "Times,18,-1,0,90,1,0,0,0,0";
    qproperty-styleFont2: "Times,12,-1,0,90,0,0,0,0,0";
    qproperty-styleFont3: "Times,12,-1,0,90,0,0,0,0,0";
    qproperty-styleFont4: "Times,12,-1,0,90,0,0,0,0,0";
    qproperty-styleFontMargin: "Times,14,-1,0,90,0,0,0,0,0";
}
```

For gcodeDisplay widget's default G-code lexer:

- styleColor0 = Default = digit characters
- styleColor1 = Comments = characters inside of *msg()*
- styleColor2 = Key = alphabetic characters
- styleColor3 = Assignment = (*%*, *<*, *>*, *#*, *=*)
- styleColor4 = Value = (*[*, *]*)

Font definitions: "style name, size, -1, 0, bold setting (0-99), italics (0-1), underline (0-1),0,0,0"

It is based on PyQt's QsciScintilla

This is an extension of the gcodeDisplay widget that adds editing convenience.

It is based on PyQt's QWidget which incorporates GcodeDisplay widget

This Displays the current G-code in a graphical form.

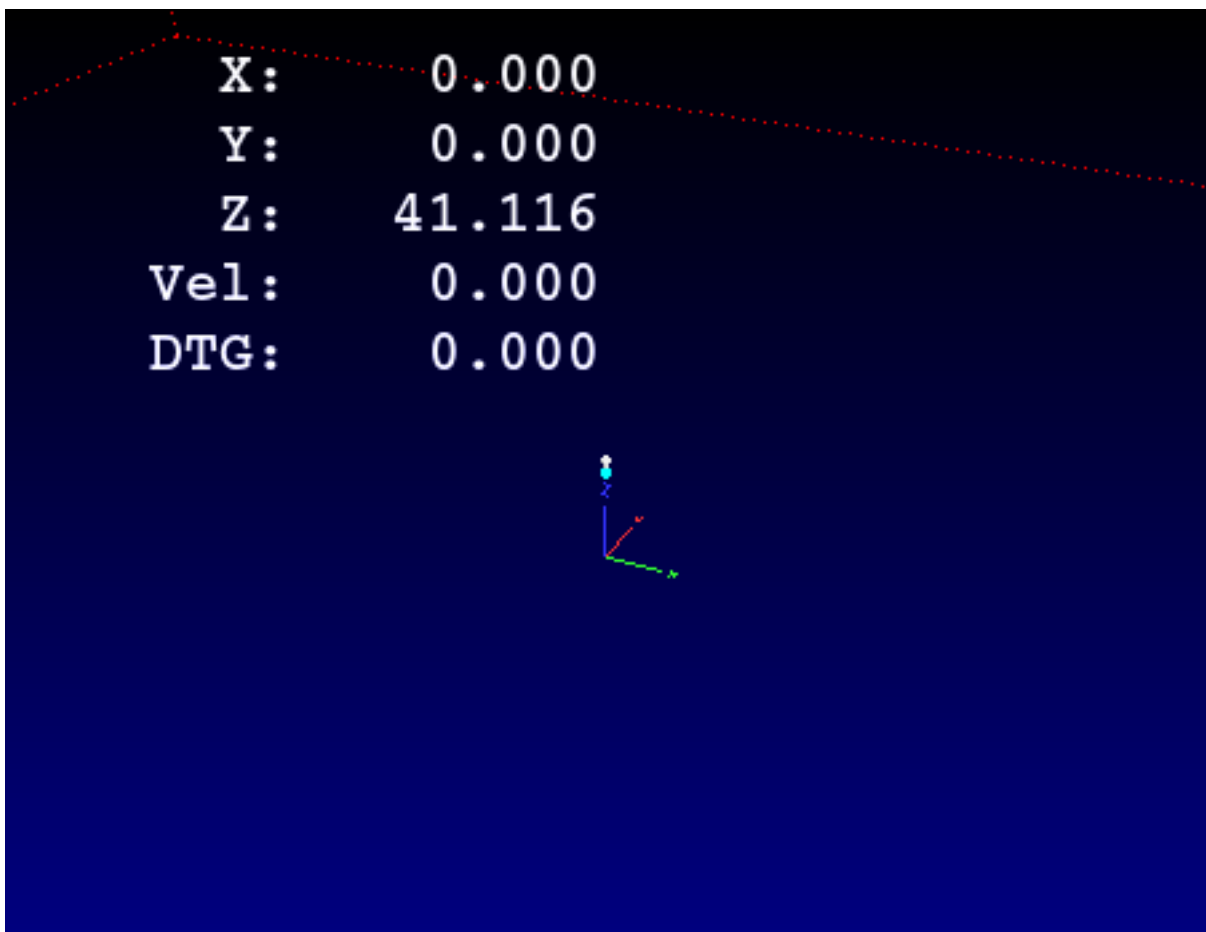


Figure 10.83: Graphics Display

Properties that can be set via stylesheets:

- *_view*
- *_dro*
- *_dtg*

- *_metric*
 - *_overlay*
 - *_offsets*
 - *_small_origin*
 - *overlay_color*
 - *background_color*
 - *_use_gradient_background*
 - *jog_color*
 - *Feed_color*
 - *Rapid_color*
 - *InhibitControls*
 - *MouseButtonMode*
 - *MouseWheelInvertZoom*
-

_view Expects a string. The following shows an example of how to set this property:

```
#gcodegraphics{
qproperty-_view:z;}
```

Setting this property sets the default view on GUI load. Valid choices for a lathe are p, y, y2. For other screens, valid choices are p, x, y, z, z2.

_dro Expects a boolean. The following shows an example of how to set this property:

```
#gcodegraphics{
qproperty-_dro:False;}
```

Setting this property will determine whether or not to show the DRO.

_dtg Expects a boolean. The following shows an example of how to set this property:

```
#gcodegraphics{
qproperty-_dtg:False;}
```

Setting this property will determine whether or not to show the Distance To Go.

_metric Expects a boolean. The following shows an example of how to set this property:

```
#gcodegraphics{
qproperty-_metric:False;}
```

Setting this property will determine whether or not to show the units in metric by default.

_overlay Expects a boolean. The following shows an example of how to set this property:

```
#gcodegraphics{
qproperty-_overlay:False;}
```

Setting this property will determine whether or not to show the overlay by default.

_offsets Expects a boolean. The following shows an example of how to set this property:

```
#gcodegraphics{
qproperty-_offsets:False;}
```

Setting this property will determine whether or not to show the offsets by default.

_small_origin Expects a boolean. The following shows an example of how to set this property:

```
#gcodegraphics{
qproperty-_small_origin:False;}
```

Setting this property will determine whether or not to show the small origin by default.

overlay_color Expects a primary, secondary, or RGBA formatted color. The following shows an example of how to set this property:

```
#gcodegraphics{
qproperty-overlay_color:blue;}
```

Setting this property will set the default overlay color.

background_color Expects a primary, secondary, or RGBA formatted color. The following shows an example of how to set this property:

```
#gcodegraphics{
qproperty-background_color:blue;}
```

Setting this property will set the default background color.

_use_gradient_background Expects a boolean. The following shows an example of how to set this property:

```
#gcodegraphics{
qproperty-_use_gradient_background:False;}
```

Setting this property will determine whether or not use a gradient background by default.

jog_color Expects a primary, secondary, or RGBA formatted color. The following shows an example of how to set this property:

```
#gcodegraphics{
qproperty-jog_color:red;}
```

Setting this property will set the default jog color.

Feed_color Expects a primary, secondary, or RGBA formatted color. The following shows an example of how to set this property:

```
#gcodegraphics{
qproperty-Feed_color:green;}
```

Setting this property will set the default feed color.

Rapid_color Expects a primary, secondary, or RGBA formatted color. The following shows an example of how to set this property:

```
#gcodegraphics{
qproperty-Rapid_color: rgba(0, 0, 255, .5);}
```

Setting this property will set the default rapid color.

InhibitControls Expects a boolean. The following shows an example of how to set this property:

```
#gcodegraphics{
qproperty-InhibitControls:True;}
```

Setting this property will determine whether or not to inhibit external controls by default.

MouseButtonMode Expects an integer. The following shows an example of how to set this property:

```
#gcodegraphics{
qproperty-MouseButtonMode:1;}
```

Setting this property changes the button behavior of the mouse to rotate, move or zoom within the preview.

There are 12 valid modes:

Mode	Move	Zoom	Rotate
0	Left	Middle	Right
1	Middle	Right	Left
2	Middle	Left	Right
3	Left	Right	Middle
4	Right	Left	Middle
5	Right	Middle	Left

Modes 6-11 are intended for Machines that only require a 2D preview such as plasma or some Lathes and have no rotate button assigned.

Mode	Move	Zoom
6	Left	Middle
7	Middle	Left
8	Right	Left
9	Left	Right
10	Middle	Right
11	Right	Middle

MouseWheelInvertZoom Expects a boolean. The following shows an example of how to set this property:

```
#gcodegraphics{
qproperty-MouseWheelInvertZoom:True;}
```

Setting this property to True will invert the zoom direction when zooming with the mouse wheel. The ACTION library can control the G-code graphics widget.

- ACTION.RELOAD_DISPLAY() - reload the current program which recalculates the origin/offsets.
- ACTION.SET_GRAPHICS_VIEW(view) The following commands can be sent:

```
clear
zoom-in
zoom-out
pan-up
pan-down
pan-right
pan-left
rotate-cw
rotate-ccw
rotate-up
rotate-down
overlay-dro-on
overlay-dro-off
overlay-offsets-on
overlay-offsets-off
alpha-mode-on
alpha-mode-off
inhibit-selection-on
inhibit-selection-off
dimensions-on
dimensions-off
grid-size
record-view
```

```
set-recorded-view
P
X
Y
Y2
Z
Z2
```

- `ACTION.ADJUST_PAN(X,Y)` - directly set the relative pan of view in x and y direction
- `ACTION.ADJUST_ROTATE(X,Y)` - directly set the relative rotation of view in x and y direction

It is based on PyQt's opengl widget.

10.4.7.3 StateLabel Widget

This will display a label based on true/false states of the machine controller. You can select different text based on true or false. These states are selectable via these properties:

- *css_mode_status*
When true machine is in G96 Constant Surface Speed Mode
- *diameter_mode_status*
When true machine is in G7 Lathe Diameter Mode
- *fpr_mode_status*
When true machine is in G95 Feed per revolution Mode
- *metric_mode_status*
When true machine is in G21 Metric Mode

Other Properties:

- *true_textTemplate*
This will be the text set when the option is true. You can use Qt rich text code for different fonts/-colours etc. Typical template for metric mode in true state, might be: *Metric Mode*
- *false_textTemplate*
This will be the text set when the option is false. You can use Qt rich text code for different fonts/-colours etc. Typical template for metric mode in false state, might be: *Imperial Mode*

It is based on PyQt's QLabel.

This will display a label based on variable states of the machine controller. You can change how the state will be display by substituting You can use Rich text for different fonts/colors etc. These states are selectable:

- *actual_spindle_speed_status*
Used to display the actual spindle speed as reported from the HAL pin spindle.0.speed-i It's converted to RPM. Typically would use a textTemplate of %d
- *actual_surface_speed_status*
Used to display the actual cutting surface speed on a lathe based on X axis and spindle speed. It's converted to distance per minute. Typically would use a textTemplate of %4.1f (feet per minute) and altTextTemplate of %d (meters per minute)
- *blendcode_status*
Shows the current g64 setting

- *current_feedrate_status*
Shows the current actual feedrate
 - *current_FPU_status*
Shows the current actual feed per unit
 - *fcode_status*
Shows the current programmed F Code setting
 - *feed_override_status*
Shows the current feed override setting in percent
 - *filename_status*
Shows the last loaded file name
 - *filepath_status*
Shows the last loaded full file path name
 - *gcode_status*
Shows all active G-codes
 - *gcode_selected_status*
Show the current selected G-code line
 - *halpin_status*
Shows the HAL pin output of a selected HAL pin
 - *jograte_status*
Shows the current QtVCP based Jog Rate
 - *jograte_angular_status*
Shows the current QtVCP based Angular Jog Rate
 - *jogincr_status*
Shows the current QtVCP based Jog increment
 - *jogincr_angular_status*
Shows the current QtVCP based Angular Jog increment
 - *machine_state_status*
Shows the current machine interpreter state using the text described from the *state_list*.
The interpreter states are: Estopped, Running, Stopped, Paused, Waiting, Reading
 - *max_velocity_override_status*
Shows the current max axis velocity override setting
 - *mcode_status*
Shows all active M-codes
 - *requested_spindle_speed_status*
Shows the requested spindle speed - actual may be different.
 - *rapid_override_status*
Shows the current rapid override setting in (0-100) percent
 - *spindle_override_status*
Shows the current spindle override setting in percent
 - *timestamp_status*
Shows the time based on the system settings.
An example of a useful textTemplate setting: `%I:%M:%S %p` see the python time module for more info
-

- *tool_comment_status*
returns the comment text from the current loaded tool
- *tool_diameter_status*
returns the diameter from the current loaded tool
- *tool_number_status*
returns the tool number of the current loaded tool
- *tool_offset_status*
returns the offset of the current loaded tool, indexed by *index_number* to select axis (0=x,1=y,etc)
- *user_system_status*
Shows the active user coordinate system (G5x setting)

Other Properties:

- *index_number*
Integer that specifies the tool status index to display.
- *state_label_list*
List of labels used for different machine states.
- *halpin_names*
Name of the halpin to monitor (including HAL component basename).
- *textTemplate*
This uses python formatting rules to set the text output. This is usually used for imperial (G20) or angular numerical settings, though not every option has imperial/metric conversion. One can use %s for no conversion, %d for integer conversion, %f for float conversion. etc You can also use Qt rich text code. Typical template used for formatting imperial float numbers to text eg. %9.4f or %9.4f inch
- *alt_textTemplate*
This uses python formatting rules to set the text output. This is usual used for metric (G21) numerical settings. Typical template used for formatting metric float to text eg. %10.3f or %10.3f mm

It is based on PyQt's QLabel

Status image switcher will switch between images based on LinuxCNC states.

- *watch spindle* would toggle between 3 images (stop, fwd, revs)
- *watch axis homed* would toggle between 2 images (axis not homed, axis homed)
- *watch all homed* would toggle between 2 images (not all homed, all homed)
- *watch hard limits* would toggle between 2 images or one per joint

Here is an example of using it to display an icon of Z axis homing state:



In the properties section notice that:

- *watch axis homed* is checked
- *axis letter* is set to Z

If you double click the *image list* a dialog will show and allow you to add image paths to. If you have one image as an icon and one clear image then that will look like it shows and hides the icon.

Selecting image paths can be done by selecting the *pixmap* property and selecting an image. Note: The pixmap setting is for test display only and will be ignored outside of Designer. Right click the image name and you should see *copy path* Click *copy path* Now double click the *image list* property so the dialog shows. Click the *New* button Paste the image path in the entry box Do that again for the next image - use a clear image to represent a hidden icon.

You can test display the images from the image list by changing the *image number* In this case 0 is unhomed 1 would be homed This is for test display only and will be ignored outside of Designer.

This widget displays one of three panels based on LinuxCNC's mode. This allows you to automatically display different widgets on Manual, MDI and Auto modes.

todo It is based on PyQt's QStacked widget.

This widget allows the user to select jog increment values for jogging. The jogging values come from the INI file under: *[DISPLAY], INCREMENTS* or *[DISPLAY], ANGULAR_INCREMENTS* This will be available to all widgets through STATUS. You can select linear or angular increments by the property *linear_option* in Designer property editor.

It is based on PyQt's combobox

This widget doesn't add anything visually to a screen but sets up important options. This is the preferred way to use these options

These properties can be set in designer, in python handler code or (if appropriate) in stylesheets.

These include:

- *halCompBaseName:*
If left empty QtVCP will use the screen's name as the HAL component's basename. If set, QtVCP will use this string as the HAL component's basename. If the -c command line option is used when loading QtVCP, QtVCP will use the name specified in the command line - it overrides all above options. If you programmatically set the basename in the handlerfile - it will override all above options. This option cannot be set in stylesheets.
 - *notify_option:*
Hooking into the desktop notification bubbles for error and messages
 - *notify_max_messages:*
Number of messages shown on screen at one time.
 - *catch_close_option:*
Catching the close event to pop up a *are you sure* prompt
 - *close_overlay_color:*
Color of transparent layer shown when quitting.
 - *catch_error_option:*
monitoring the LinuxCNC error channel. This also sends the message through STATUS to anything that registers
 - *play_sounds_option:*
playing sounds using *beep*, *espeak* and the system sound
 - *use_pref_file_option:*
setting up a preference filepath. Using the magic word *WORKINGFOLDER* in the preference file path will be replaced with the launched configuration path ie. *WORKINFOLDER/my_preferences*
 - *use_send_zmq_option:*
Used to initiate ZMQ based outgoing messages.
 - *use_receive_zmq_messages:*
Used to initiate ZMQ based in coming messages. These messages can be used to call functions in the handler file. Allowing external programs to intergrate tightly with qtvcp based screens.
 - *embedded_program_option:*
Embed programs defined in the INI.
 - *default_emebed_tab*
This is the property for a default location to embed external programs. It would be set to the designer name of a tab page widget.
 - *focusOverlay_option:*
Focus_overlay will put a transparent image or colored panel over the main screen to emphasize focus to an external event - typically a dialog.
 - *messageDialog_option:*
sets up the message dialog - used for general messages
 - *message_overlay_color:*
Color of transparent layer shown when the message dialog is shown.
 - *closeDialog_option:*
sets up the standard close screen prompt dialog
 - *entryDialog_option:*
sets up the numerical entry dialog
 - *entryDialogSoftKey_option:*
sets up a floating software keyboard when entry dialog is focused.
-

- *entry_overlay_color*:
Color of transparent layer shown when the entry dialog is shown.
 - *toolDialog_option*:
sets up the manual tool change dialog, including HAL pin.
 - *tool_overlay_color*:
Color of transparent layer shown when the tool dialog is shown.
 - *ToolUseDesktopNotify*:
option to use desktop notify dialogs for manual tool change dialog.
 - *ToolFramesless*:
Framesless dialogs can not be easily moved by users.
 - *fileDialog_option*:
sets up the file choosing dialog.
 - *file_overlay_color*:
Color of transparent layer shown when the file dialog is shown.
 - *keyboardDialog_option*:
sets up a keyboard entry widget.
 - *keyboard_overlay_color*:
Color of transparent layer shown when the keyboard dialog is shown.
 - *versaProbe_option*:
sets up the versa style probe dialog
 - *versaProbe_overlay_color*:
Color of transparent layer shown when the versaProbe dialog is shown.
 - *macroTabDialog_option*:
sets up the macro selection dialog
 - *macroTab_overlay_color*:
Color of transparent layer shown when the macroTab dialog is shown.
 - *camViewDialog_option*:
sets up the camera alignment dialog
 - *camView_overlay_color*:
Color of transparent layer shown when the camView dialog is shown.
 - *toolOffset_option*:
sets up the tool offset display/editor dialog
 - *toolOffset_overlay_color*:
Color of transparent layer shown when the toolOffset dialog is shown.
 - *originOffset_option*:
sets up the origin display/editor dialog
 - *originOffset_overlay_color*:
Color of transparent layer shown when the originOffset dialog is shown.
 - *calculatorDialog_option*:
sets up the calculatory entry dialog
-

- *calculator_overlay_color*:
Color of transparent layer shown when the calculator dialog is shown.
- *machineLogDialog_option*:
sets up a dialog to display logs from the machine and qtvcp
- *machineLog_overlay_color*:
Color of transparent layer shown when the machineLog dialog is shown.
- *runFromLineDialog_option*:
sets up a dialog to display starting options when starting machine execution from a arbitrary line.
- *runFromLine_overlay_color*:
Color of transparent layer shown when the runFromLine dialog is shown.

The screen designer chooses the default settings of the screenOptions widget. Once chosen, most won't ever need to be changed. but if needed some can be changed in the handler file or in stylesheets. Some settings are only checked on startup so will not cause changes after startup. In these cases you would need to make the changes in Qt designer only.

ie. in the handler file Here we reference the widget by the QtDesigner user defined name:

```
# red,green,blue,alpha 0-255
color = QtGui.QColor(0, 255, 0, 191)
self.w.screen_options.setProperty('close_overlay_color', color)
self.w.screen_options.setProperty('play_sounds_option',False)
```

ie. In style sheets Here we can reference the widget by QtDesigner user defined name or by widget class name.

```
/* red, green, blue 0-255, alpha 0-100% or 0.0 to 1.0 */
/* the # sign is used to refer to QtDesigner defined widget name */
/* matches/applied to only this named widget */

#screen_options {
qproperty-close_overlay_color: rgba(0, 255, 0, 0.75) }

/* red, green, blue 0-255, alpha 0-100% or 0.0 to 1.0 */
/* use widget class name - matches/applied to all widgets of this class*/

ScreenOptions {
qproperty-close_overlay_color: rgba(0, 255, 0, 0.75) }
```

If the preference file option is selected, screenOption widget will make an INI based preference file. While other QtVCP widgets will add to this list, the screenOptions widget will add these entries:

Under the heading: *SCREEN_OPTIONS*:

- *catch_errors* = -True or False
- *desktop_notify* = -True or False (whether to display errors/messages in the system's notification mechanism)
- *notify_max_msgs* = -Integer (number of displayed errors at one time)
- *shutdown_check* = -True or False (whether to pop a confirmation dialog)
- *sound_player_on* = -True or False (turns all sounds on or off)

Under the heading: *MCH_MSG_OPTIONS*

- *mchnMsg_play_sound* = -True or False (to play alert sound when dialog pops)
- *mchnMsg_speak_errors* = -True or False (to use Espeak to speak error messages)
- *mchnMsg_speak_text* = -True or False (to use Espeak to speak all other messages)
- *mchnMsg_sound_type* = -sound to play when messages displayed

Under the heading: *USER_MSG_OPTIONS*

- *usermsg_play_sound* = -True or False (to play alert sound when dialog pops)
- *userMsg_sound_type* = -sound to play when user messages displayed
- *userMsg_use_focusOverlay* = -True or False

Under the heading: *SHUTDOWN_OPTIONS*

- *shutdown_play_sound* = -True or False
- *shutdown_alert_sound_type* = -sound to play when messages displayed
- *shutdown_exit_sound_type* = -sound to play when messages displayed
- *shutdown_msg_title* = -Short title string to display in dialog
- *shutdown_msg_focus_text* = -Large text string to superimpose in focus layer
- *shutdown_msg_detail* = -Longer descriptive string to display in dialog

Under the heading: *NOTIFY_OPTIONS*

- *notify_start_greeting* = - True or False (whether to display a greeting dialog on start up)
- *notify_start_title* = - Short Title string. If the speak option is also selected it will be spoken with Espeak.
- *notify_start_detail* = - Longer description string.
- *notify_start_timeout* = - time in seconds to display before closing.

Note

In Debian/Ubuntu/Mint based systems these sounds should be available as sound-type entries above:
(These Sound options require python3-gst1.0 installed.)

- ERROR
 - READY
 - DONE
 - ATTENTION
 - RING
 - LOGIN
 - LOGOUT
 - BELL
-

Note

You can also specify a file path to an arbitrary audio file. (You can use ~ in path to substitute for the user home file path)

Note

If the Beep kernel module is installed and it is not disabled, these sound-type entries are available:

- BEEP
 - BEEP_RING
 - BEEP_START
-

Note

If the Espeak module (python3-espeak) is install you can use the entry *SPEAK* to pronounce text:

- *SPEAK my message*
-

This widget allow the user to adjust LinuxCNC setting via a slide.

The widget can adjust:

- Jog rate
- Angular jog rate
- Feed rate
- spindle override rate
- Rapid override rate

StatusSlider has properties that can be set in designer, in python handler code or (if appropriate) in stylesheets.

- `halpin_option` - sets option to make a HAL float pin that reflects current value.
- `rapid_rate` - selects a rapid override rate slider
- `feed_rate` - selects a feed override rate slider
- `spindle_rate` - selects a spindle override rate slider
- `jograte_rate` - selects a linear jograte slider
- `jograte_angular_rate` - selects a angular jograte slider
- `max_velocity_rate` - selects a maximum velocity rate slider
- `alertState` - a string to define style change. (read-only - *under*, *over* and *normal*)
- `alertUnder` - set the float value that signals the stylesheet for *under* warning.
- `alertOver` - set the float value that signals the stylesheet for *over* warning.

ie. in handler file:

```
self.w.status_slider.setProperty('spindle_rate',True)
self.w.status_slider.setProperty('alertUnder',35)
self.w.status_slider.setProperty('alertOver',100)
```

ie. In style sheets:

```
/* warning colors for overrides if out of normal range*/
/* widget object name is slider_spindle_ovr */

    #slider_spindle_ovr[alertState='over'] {
        background: red;
    }
    #slider_spindle_ovr[alertState='under'] {
        background: yellow;
    }
```

It is based on PyQt's QSlider

This widget gives status on the selected LinuxCNC state.

The state options are:

- is_paused_status
- is_estopped_status
- is_on_status
- is_idle_status_
- is_homed_status
- is_flood_status
- is_mist_status
- is_block_delete_status
- is_optional_stop_status
- is_joint_homed_status
- is_limits_overridden_status
- is_manual_status
- is_mdi_status
- is_auto_status
- is_spindle_stopped_status
- is_spindle_fwd_status
- is_spindle_rev_status
- is_spindle_at_speed_status

There are properties that can be changed:

- halpin_option - Adds an output pin that reflects selected state
- invert_state_status - Invert the LED state compared to the LinuxCNC state.
- diameter -Diameter of the LED

- color - Color of the LED when on.
- off_color - Color of the LED when off.
- alignment - Qt Aliment hint.
- state - Current state of LED (for testing in designer)
- flashing - Turns flashing option on and off.
- flashRate - Sets the flash rate.

The LED properties can be defined in a stylesheet with the following code added to the .qss file. The name_of_led would be the name defined Designer's editor.

```
State_LED #name_of_led{
qproperty-color: red;
qproperty-diameter: 20;
qproperty-flashRate: 150;
}
```

It is based on the LED widget

This widget allows setting values using buttons while displaying a bar. It also has an optional hi/low toggle button that can be held down to set the levels.

The widget can adjust:

- Jog rate
- angular jog rate
- Feed rate
- Spindle override rate
- Rapid override rate

It is based on PyQt's QProgressBar

This widget allows you to manually select a user system by pressing and holding. If you don't set the button text it will automatically update to the current system.

It is based on PyQt's QToolButton

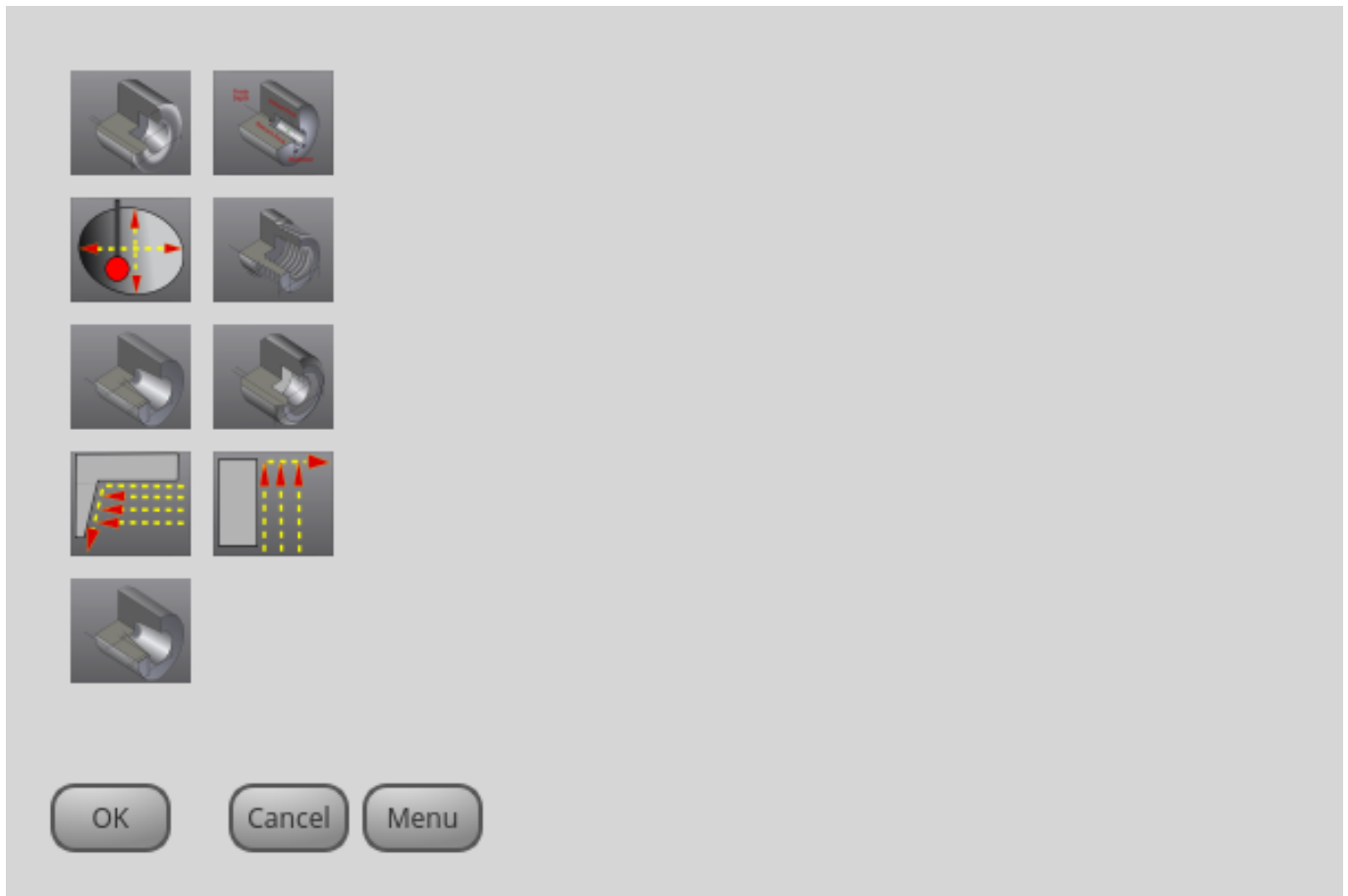


Figure 10.84: Macrotab

This Widget allows a user to select and adjust special macro programs for doing small jobs. It uses images for visual representation of the macro and for an icon. It searches for special macros using the INI definition:

```
[RS274NGC]
SUBROUTINE_PATH =
```

The macros are Oword subroutine with special comments to work with the launcher. The first three lines must have the keywords: (The forth is optional) Here is a sample for the first four lines in an Oword file:

```
; MACROCOMMAND=Entry1,Entry2
; MACRODEFAULTS=0,true
; MACROIMAGE=my_image.svg,Icon layer number, Macro layer number
; MACROOPTIONS=load:yes,save:yes,default:default.txt,path:~/macros
```

This is the first line in the Oword file. It is a comma separated list of text to display above an entry. There will be one for every variable required in the Oword function. If the macro does not require variables, only add ; *MACROCOMMAND=*

This must be the second line in the Oword file. It is a comma separated list of the default values for each variable in the Oword function. If you use the word *true* or *false* in the list, a checkbox will be shown.

This must be the third line in the Oword file. if using a SVG image file, the must end b .svg The image must be added to an svg layer. It uses layers to define different images for macro and icon. The first

entry will be the SVG image file name. It is assumed to be in the same folder as the Oword file. The second item will be the image layer. the optional third entry will be the icon layer. If the third entry is missing, the same image will be used for macro and icon.

If using a png/jpg image file. The first entry is the image filename. It is assumed the image file are in the same folder as the macro. The optional second entry will be the icon filename. If the second entry is missing the same image will be used for macro and image.

If the keyword is present but the entries are missing , no images will be used.

This optional line must be the forth line in the Oword file. It is a comma separated list of keyword and data.

- *LOAD:yes* - show a load button
- *SAVE:yes* -show a save button

One can enter MDI commands here. A popup keyboard is available. There are also embedded commands available from this Widget. Enter any of these case sensitive commands to load the respective program or access the feature:

- HALMETER - Starts LinuxCNC utility [halmeter](#)
- HALSHOW - Starts LinuxCNC utility [halshow](#)
- HALSCOPE - Starts LinuxCNC utility [halscope](#)
- STATUS - Starts LinuxCNC utility [status](#)
- CALIBRATION - Starts LinuxCNC utility [calibration](#)
- CLASSICLADDER - Starts the [ClassicLadder GUI](#) if the ClassicLadder realtime HAL component was loaded by the machine's config files
- PREFERENCE - Loads the preference file onto the gcodeEditor
- CLEAR HISTORY - Clears the MDI History
- net - See [halcmd net COMMAND](#). An error will result if the command is unsuccessful.
 - *Syntax:* net <signal name> <pin name>
 - *Example:* net plasmac:jog-inhibit motion.jog-stop
- setp - Sets the value of a pin or a parameter. Valid values depend on the object type of the pin or parameter. An error will result if the data types do not match or the pin is connected to a signal.
 - *Syntax:* setp <pin/parameter-name> <value>
 - *Example:* setp plasmac.resolution 100
- unlinkp - Disconnects a pin from a signal. An error will result if the pin does not exist. Running LinuxCNC from terminal may help determine the root cause as error messages from hal_lib.c will be displayed there.
 - *Syntax:* unlinkp <pin name>
 - *Example:* unlinkp motion.jog-stop

Note that the MDIline function "spindle_inhibit" can be used by a GUI's handler file to inhibit M3, M4, and M5 spindle commands if necessary.

It is based on PyQt's QLineEdit.

Displays a scrollable list of past MDI command. A edit line is embedded for MDI commands. The same MDIline embedded commands may be accessed from this Widget.

The history is recorded in a file defined in the INI. under the heading [DISPLAY] (this shows the default)

```
MDI_HISTORY_FILE = '~/axis_mdi_history'
```

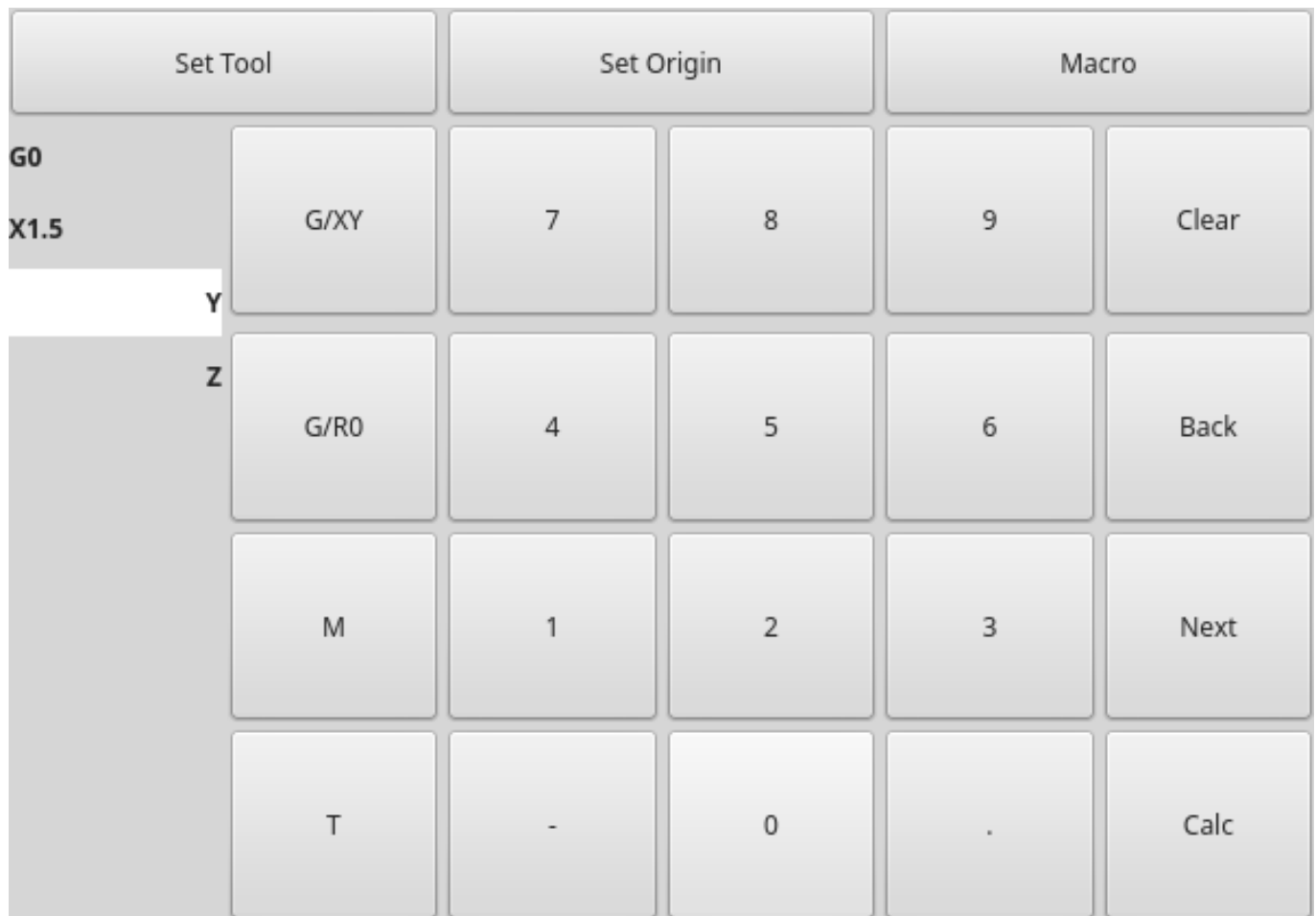


Figure 10.85: MDI Touchy

This widget display button and entry lines for use with entering MDI commands. It is based on Linuxcnc's Touchy screen's MDI entry process. It's large buttons are most useful for touch screens.

To use MDITouchy, first press one of the *G/XY*, *G/RO*, *M* or *T* button. On the left, will show the current line that can be filled out, then press *Next* for the next line. *Calc* will pop up a calculator dialog. *Clear* clears the current entry. *Back* allows you to change previous line entries.

The widget requires an explicit call to MDITouchy's python code to actually run the MDI command. For handler file code: if the widget was named `mditouchy` in designer, this command would run the displayed MDI command.

```
self.w.mditouchy.run_command()
```

For action button use: if the widget was named `mditouchy` in designer, use the action button's *Call python commands* option and enter:

```
INSTANCE.mditouchy.run_command()
```

The macro button will cycle through macro's defined in the INI heading `[DISPLAY]` add one or more `'MACRO = '` lines. Each should be of the format:


```
MACRO = increment xinc yinc
```

In this example, increment is the name of the macro, and it accepts two parameters, named xinc and yinc.

Now, place the macro in a file named *increment.ngc*, in the *PROGRAM_PREFIX* directory or any directory in the *SUBROUTINE_PATH*. (specified in the INI file)

It should look like:

```
O<increment> sub
G91 G0 X#1 Y#2
G90
O<increment> endsub
```

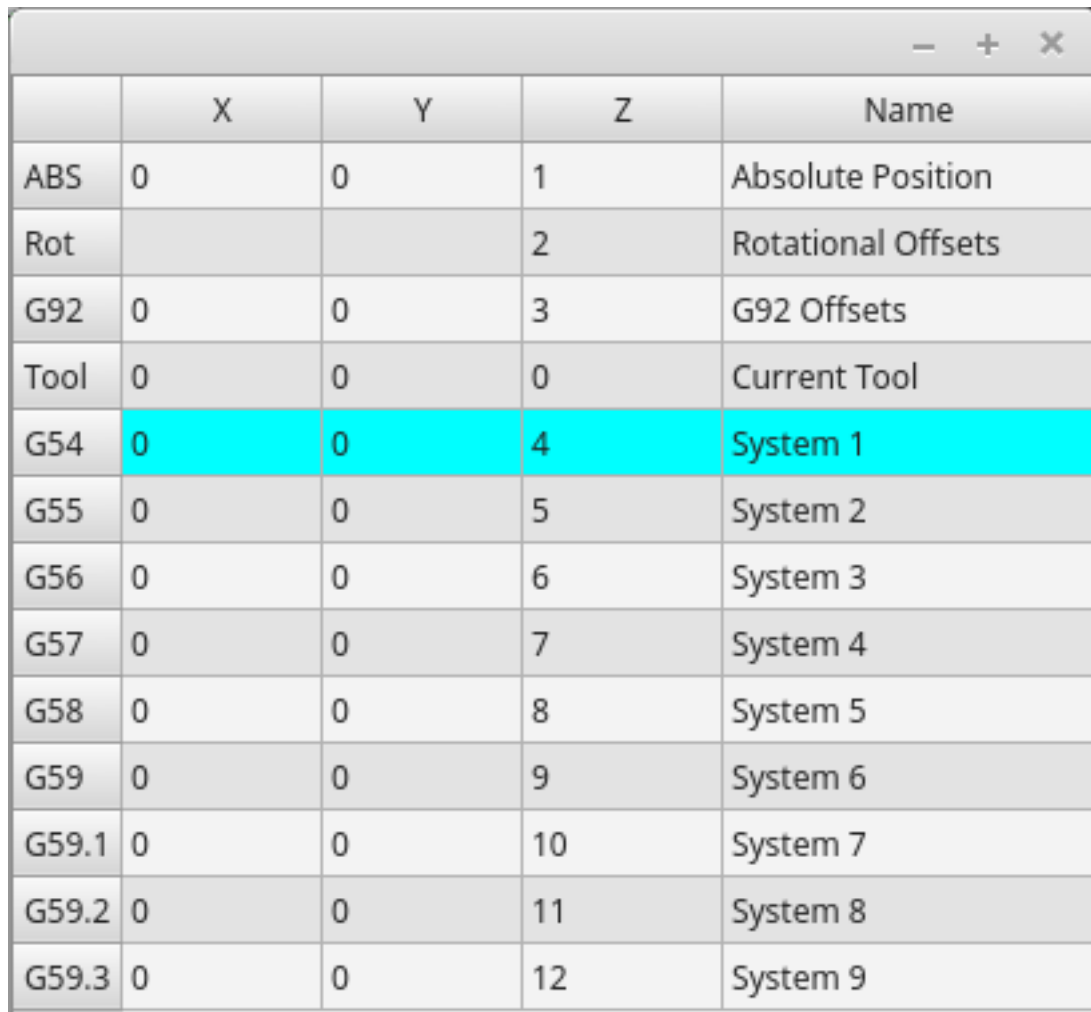
Notice the name of the sub matches the file name and macro name exactly, including case.

When you invoke the macro by pressing the Macro button you can enter values for xinc and yinc. These are passed to the macro as *#1* and *#2* respectively. Parameters you leave empty are passed as value 0.

If there are several different macros, press the Macro button repeatedly to cycle through them.

In this simple example, if you enter -1 for xinc and invoke the running of the MDI cycle, a rapid *G0* move will be invoked, moving one unit to the left.

This macro capability is useful for edge/hole probing and other setup tasks, as well as perhaps hole milling or other simple operations that can be done from the panel without requiring specially-written G-code programs.



	X	Y	Z	Name
ABS	0	0	1	Absolute Position
Rot			2	Rotational Offsets
G92	0	0	3	G92 Offsets
Tool	0	0	0	Current Tool
G54	0	0	4	System 1
G55	0	0	5	System 2
G56	0	0	6	System 3
G57	0	0	7	System 4
G58	0	0	8	System 5
G59	0	0	9	System 6
G59.1	0	0	10	System 7
G59.2	0	0	11	System 8
G59.3	0	0	12	System 9

Figure 10.86: origin Offset View

This widget allows one to modify User System origin offsets directly. It will update LinuxCNC's Parameter file for changes made or found. The settings can only be changed in LinuxCNC after homing and when the motion controller is idle. The display and entry will change between metric and imperial based on LinuxCNC's current G20/G21 setting. The current in-use user system will be highlighted. Extra actions can be integrated to manipulate settings. These actions depend on extra code added either to a combined widget like `originoffsetview` dialog or the screens handler code. Typical actions might be *Clear Current User offsets*, *Zero X*. Clicking on the columns and rows allows one to adjust the settings. A dialog can be made to popup for data or text entry. The comments section will be recorded in the preference file.

It is based on PyQt's `QTableView`, `QAbstractTableModel`, and `ItemEditorFactory`. Properties, functions and styles of the PyQt base objects are always available.

`OriginOffsetView` has properties that can be set in designer, in python handler code or (if appropriate) in stylesheets.

- `dialog_code_string` - sets which dialog will pop up with numerical entry.
- `test_dialog_code_string` - sets which dialog will pop up with text entry.
- `metric_template` - metric numerical data format.
- `imperial_template` - imperial numerical data format.

- styleCodeHighlight - current in-use user system highlight color.

ie. in the handler file:

```
self.w.originoffsetview.setProperty('dialog_code','CALCULATOR')
self.w.originoffsetview.setProperty('metric_template','%10.3f')
```

ie. In style sheets:

```
OriginOffsetView{
    qproperty-styleColorHighlist: lightblue;
}
```

This is a container that other widgets can be placed in. It will *grey-out* (disable) the widgets inside it depending on LinuxCNC's current state. It can selectably react to:

- machine on
- interpreter idle
- estop off
- all-homed

It is based on PyQt's QGridLayout

It is based on PyQt's

It is based on PyQt's

This widget will display images based on LinuxCNC status. You can watch:

- the state of the spindle.
- the state of all homed
- the state of a certain axis homed
- the state of hard limits

It is based on PyQt's

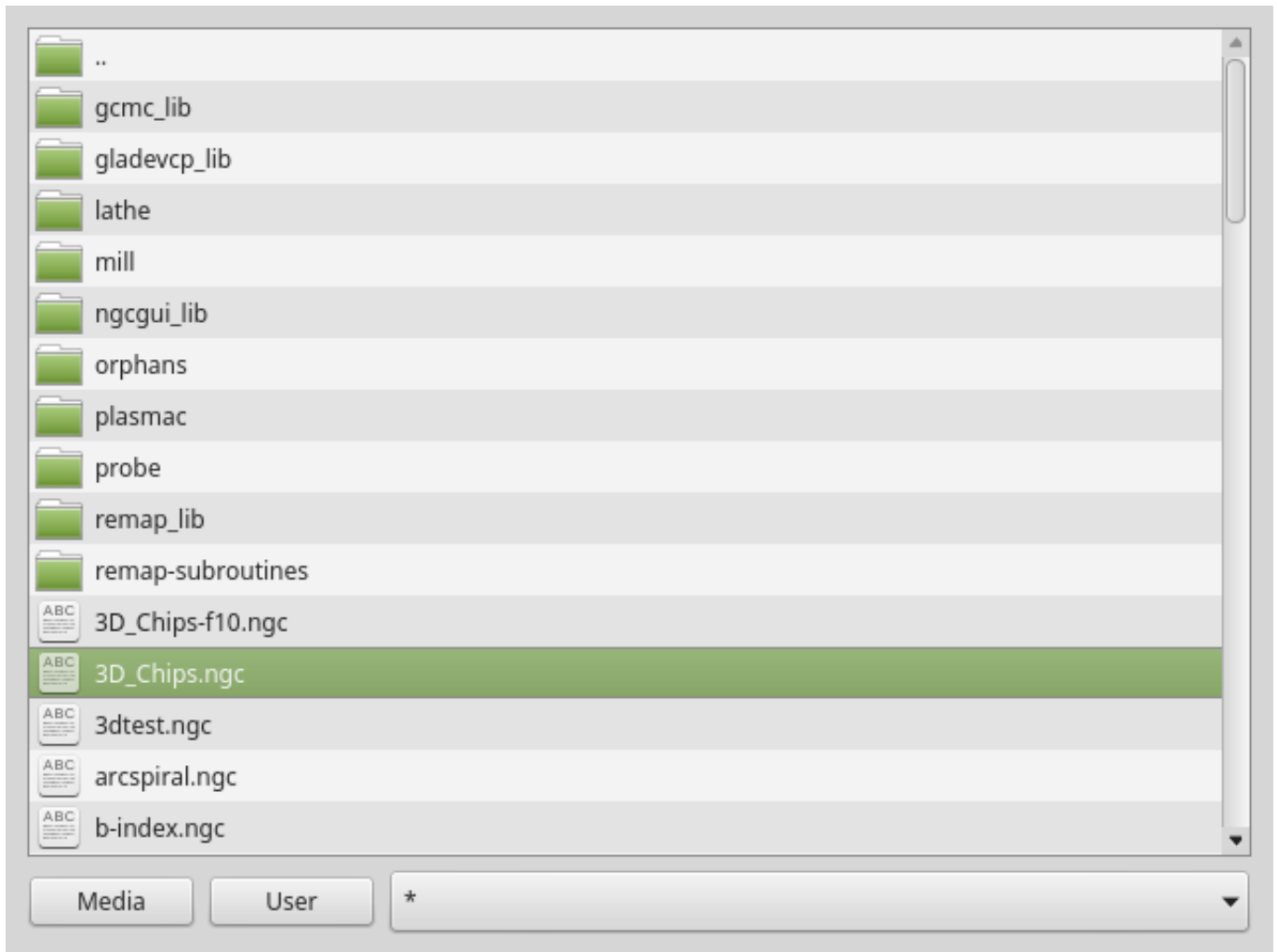


Figure 10.87: FileManager

This widget is used to select files to load. It has the ability to scroll the names with hardware such as a MPG.

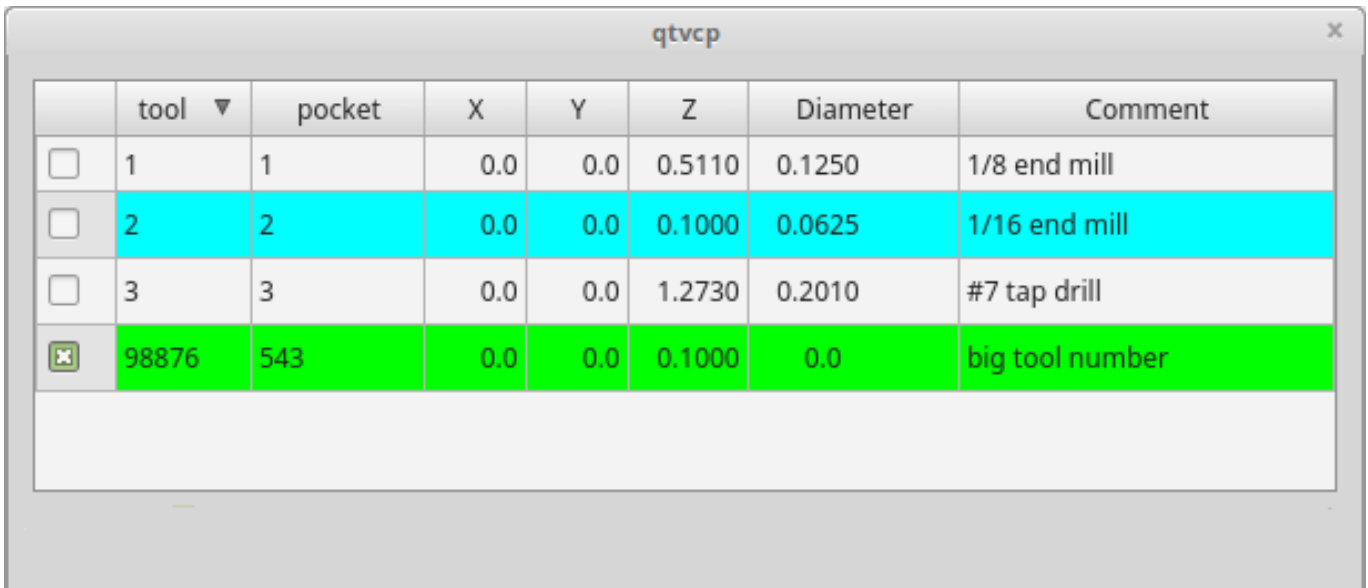
one can class patch the function *load(self,fname):* to customize file loading.

the function *getCurrentSelected()* will return a python tuple, containing the file path and whether it's a file.

```
temp = FILEMANAGER.getCurrentSelected()
print('filepath={}'.format(temp[0]))
if temp[1]:
    print('Is a file')
```

It is based on PyQt's

It is based on PyQt's



	tool ▼	pocket	X	Y	Z	Diameter	Comment
<input type="checkbox"/>	1	1	0.0	0.0	0.5110	0.1250	1/8 end mill
<input type="checkbox"/>	2	2	0.0	0.0	0.1000	0.0625	1/16 end mill
<input type="checkbox"/>	3	3	0.0	0.0	1.2730	0.2010	#7 tap drill
<input checked="" type="checkbox"/>	98876	543	0.0	0.0	0.1000	0.0	big tool number

Figure 10.88: Tool Offset View

This widget will display and allows one to modify tool offsets. It will update LinuxCNC's tool table for changes made or found. The tool settings can only be changed in LinuxCNC after homing and when the motion controller is idle. The display and entry will change between metric and imperial based on LinuxCNC's current G20/G21 setting. The current in-use tool will be highlighted. The current selected tool will be highlighted in a different color. The checkbox beside each tool can be used to select a tool(s) for an action. This action depends on extra code added either to a combined widget like `tooloffsetview` dialog or the screens handler code. Typical actions are *load selected tool*, *delete selected tools*. Clicking on the columns and rows allows one to adjust the settings. A dialog can be made to popup for data or text entry. The comments section will typically be displayed in the manual tool change dialog. If using a lathe configuration, there can be columns for X and Z wear. To use these columns to adjust the tool for wear, requires a remapped tool change routine.

It is based on PyQt's `QTableView`, `QAbstractTableModel`, and `ItemEditorFactory`. Properties, functions and styles of the PyQt base objects are always available.

`ToolOffsetView` has properties that can be set in designer, in python handler code or (if appropriate) in stylesheets.

- `dialog_code_string` - sets which dialog will pop up with numerical entry.
- `test_dialog_code_string` - sets which dialog will pop up with text entry.
- `metric_template` - metric numerical data format.
- `imperial_template` - imperial numerical data format.
- `styleCodeHighlight` - current tool-in-use highlight color.
- `styleCodeSelected` - selected highlight color

ie. In handler file:

```
self.w.tooloffsetview.setProperty('dialog_code','CALCULATOR')
self.w.tooloffsetview.setProperty('metric_template','%10.3f')
```

ie. In style sheets:

```
ToolOffsetView{
qproperty-styleColorHighlist: lightblue;
qproperty-styleColorSelected: #444;
}
```

ToolOffsetView has some function that are useful for screen builders to add actions.

- `add_tool()` - adds a blank dummy tool (99) that the user can edit to suit.
- `delete_tools()` - deletes the currently checkbox selected tools
- `get_checked_list()` - returns a list of tools selected by checkboxes.
- `set_all_unchecked()` - uncheck all selected tools.

```
self.w.tooloffsetview.add_tool()
self.w.tooloffsetview.delete_tools()
toolList = self.w.tooloffsetview.get_checked_list()
self.w.tooloffsetview.set_all_unchecked()
```

PROBING PARAMETERS	
PROBE TOOL	STEP OFF
PROBE RAPID	PROBE FEED
MAX XY DIST	XY CLEARANCE
MAX Z DIST	Z CLEARANCE
EXTRA DEPTH	EDGE WIDTH

CLEAR X	X_MINUS	0.000	X_CTR	0.000	X_PLUS	0.000	X_WIDTH	0.000
CLEAR Y	Y_MINUS	0.000	Y_CTR	0.000	Y_PLUS	0.000	Y_WIDTH	0.000
CLEAR ALL	Z	0.000	DIAM	0.000	DELTA	0.000	ANGLE	0.000

Figure 10.89: BasicProbe

Widget for probing on a mill. Used by the QtDragon screen.

10.4.7.4 Dialog Widgets

Dialogs are used to present or request immediately required information in a focused way. The typical used dialogs can be loaded using the `screenoptions` widget. You can also add them directly to the ui - but each dialog must have a unique launch name or you will see multiple dialogs displayed, one after another. You can show dialogs directly with python code but a safer way is to use STATUS messages to request the dialog to launch and to return the gathered information.

To set this up first register to catch the *general* message from STATUS:

```
STATUS.connect('general',self.return_value)
```

Add a function to call a dialog: This function must build a message DICT to send to the dialog. This message will be passed back in the general message with the addition of the RETURN variable. It is possible to add extra user information to the message. The dialog will ignore these and pass them back. *NAME* = launch code name of dialog to show. *ID* = a unique id so we process only a dialog that we requested. *TITLE* = the title to use on the dialog

```
def show_dialog(self):
    mess = {'NAME':'ENTRY','ID':'__test1__',
           'TITLE':'Test Entry'}
    ACTION.CALL_DIALOG, mess)
```

Add a callback function that processes the general message: This function should check the the name and id is the same as we sent, then it can extract the return value and any user variables. Keep in mind this function will get all general messages so the DICT keynames are not guaranteed to be there. Using the `.get()` function and or using `try/except` is advisable.

```
# process the STATUS return message
def return_value(self, w, message):
    rtn = message.get('RETURN')
    code = bool(message.get('ID') == '__test1__')
    name = bool(message.get('NAME') == 'ENTRY')
    if code and name and not rtn is None:
        print('Entry return value from {} = {}'.format(code, rtn))
```

This is a general message dialog widget. If there is an Focus Overlay widget present, it can signal it to display. If the sound library is set up it can play sounds. There are options that can be set when requesting a dialog, these would be added to the message dict.

- *TITLE*: *Attention* -Title of the dialog window
- *MESSAGE*: *your text* -Title message text in bold
- *MORE*: *your more text* - standard text under the heading
- *DETAILS*: *hidden text* - initial hidden text
- *TYPE*: *OK* - type can be *OK*, *YESNO*, *OKCANCEL*
- *ICON*: *INFO* - icon can be *QUESTION*,*INFO*,*CRITICAL*,*WARNING*
- *PINNAME* - not implemented yet
- *FOCUSTEXT*:None - text to display if focus overlay is used. Use None for no text.
- *FOCUSCOLOR*:*QColor(0, 0, 0, 150)* - color to use if focus overlay is used
- *PLAYALERT*:*'SPEAK alert!'* - sound to play if sound is available

When using STATUS's *request-dialog* function, the default launch name is *MESSAGE*. It is based on PyQt's QMessageBox.

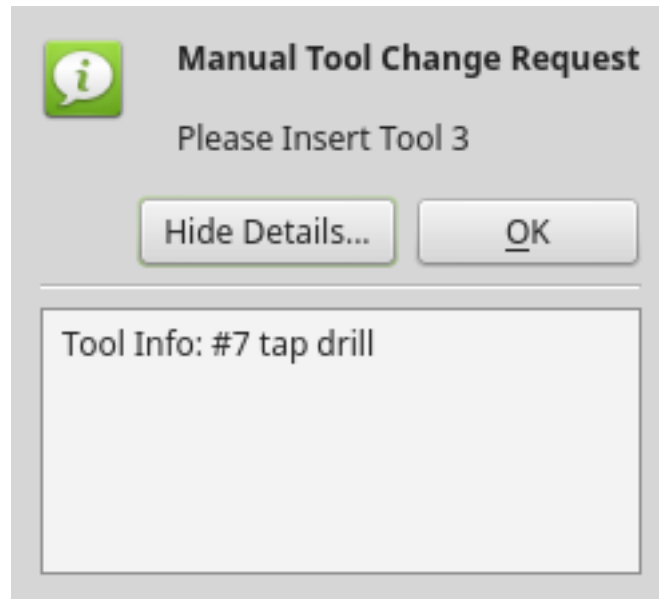


Figure 10.90: Manual Tool Change

This is used as a manual tool change prompt. It has HAL pins to connect to the machine controller. The pins are named the same as the original AXIS manual tool prompt and works the same. the tool change dialog can only be launched by HAL pins. If there is a Focus Overlay widget present, it will signal it to display.

It is based on PyQt's QMessageBox.

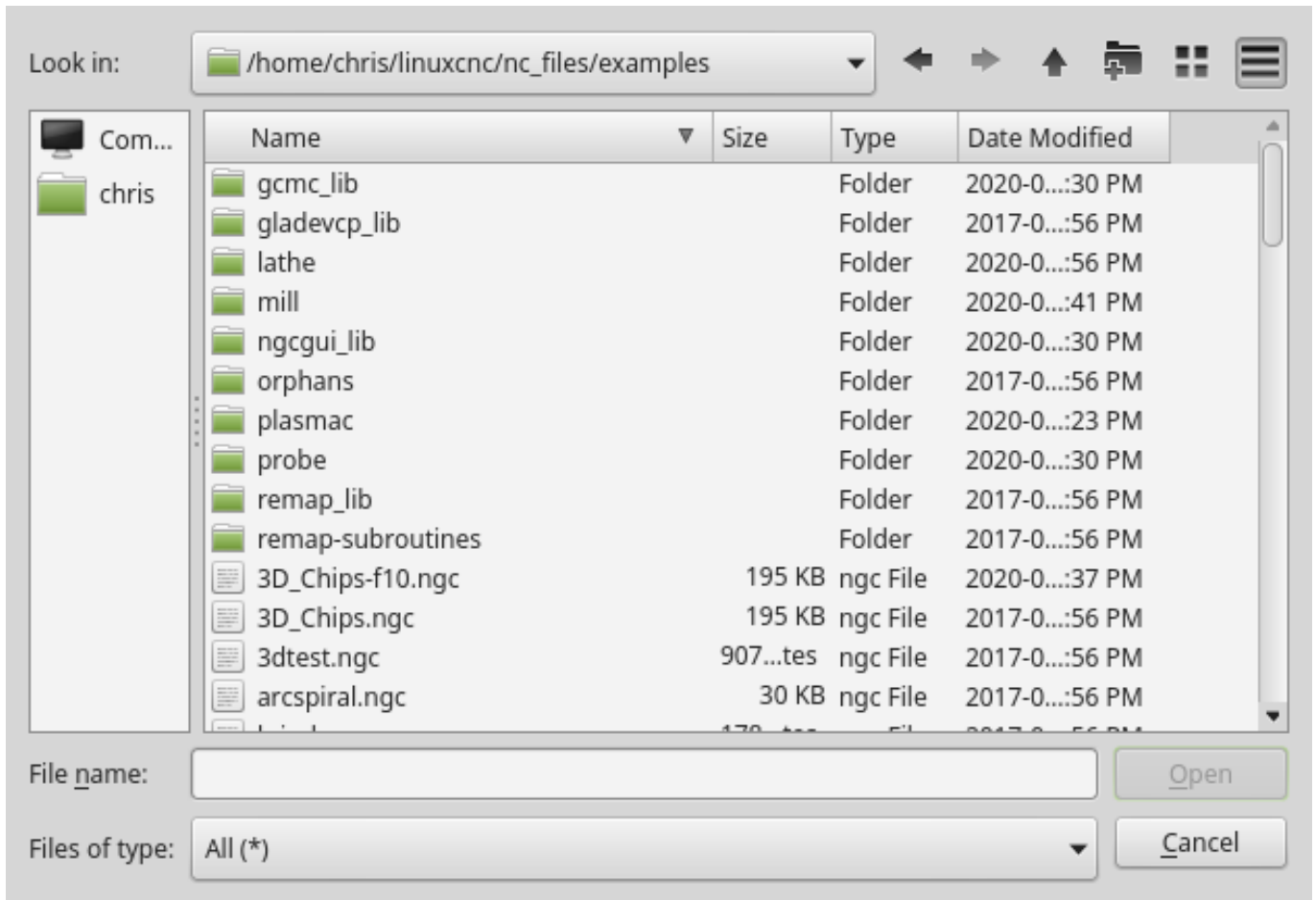


Figure 10.91: File Dialog

This is used to load G-code files. If there is a Focus Overlay widget present, it will signal it to display. When using STATUS's *request-dialog* function, the default launch names are *LOAD* or *SAVE*.

There are options that can be set when requesting a dialog, these would be added to the message dict.

- EXTENSIONS
- FILENAME
- DIRECTORY

An example python call, for a load dialog:

```
mess = { 'NAME': 'LOAD', 'ID': '_MY_DIALOG_',
         'TITLE': 'Load Some text File',
         'FILENAME': '~/linuxcnc/nc_files/someprogram.txt',
         'EXTENSIONS': 'Text Files (*.txt);;ALL Files (*.*)'
       }
ACTION.CALL_DIALOG(mess)
```

And for saving

```
mess = { 'NAME': 'SAVE', 'ID': '_MY_DIALOG_',
         'TITLE': 'Save Some text File',
```

```

    'FILENAME': '~/linuxcnc/nc_files/someprogram.txt',
    'EXTENSIONS': 'Text Files (*.txt);;ALL Files (*.*)'
}
ACTION.CALL_DIALOG(mess)

```

It is based on PyQt's QMessageBox.

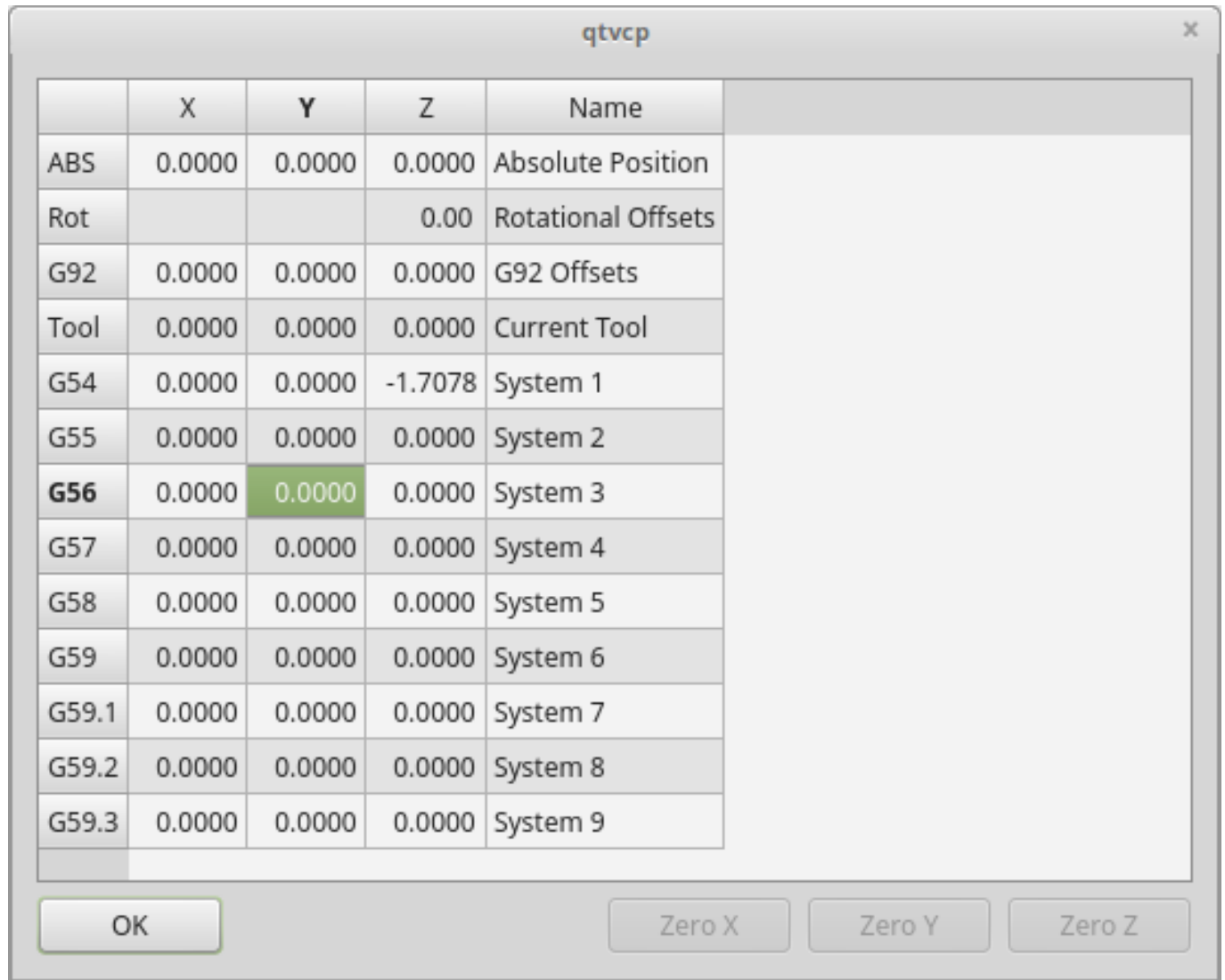


Figure 10.92: Offsets

This widget allows one to modify User System origin offsets directly. It is in a dialog form. If there is an Focus Overlay widget present, it will signal it to display. When using STATUS's *request-dialog* function, the default launch name is *ORIGINOFFSET*.

It is based on PyQt's QDialog.

Select	tool ▼	pocket	X	Y	Z	Diameter	Comment
<input type="checkbox"/>	1	1	0	0	0.5110	0.1250	1/8 end mill
<input type="checkbox"/>	2	2	0	0	0.1000	0.0625	1/16 end mill
<input type="checkbox"/>	3	3	0	0	1.2730	0.2010	#7 tap drill
<input type="checkbox"/>	99999	99999	0	0	0.1000	0	big tool number

OK
Add Tool
Delete Tool

Figure 10.93: Tool Offsets

This widget allows one to modify Tool offsets directly. It is in a dialog form. If there is an Focus Overlay widget present, it will signal it to display. When using STATUS's *request-dialog* function, the default launch name is *TOOLOFFSET*.

It is based on PyQt's QDialog.

This is a dialog for displaying the macrotab widget. Macrotab displays a choice of macro programs to run using icons. If there is a Focus Overlay widget present, it will signal it to display. When using STATUS's *request-dialog* function, the default launch name is *MACROTAB*.

This is a dialog to display the camview object for Webcam part alignment. When using STATUS's *request-dialog* function, the default launch name is *CAMVIEW*. It is based on PyQt's QDialog.

This is a dialog to display an edit line for information entry, such as origin offset. It returns the entry via STATUS messages using a python DICT. The DICT contains at minimum, the name of the dialog requested and an id code. When using STATUS's *request-dialog* function, the default launch name is *ENTRY*.

It is based on PyQt's QDialog

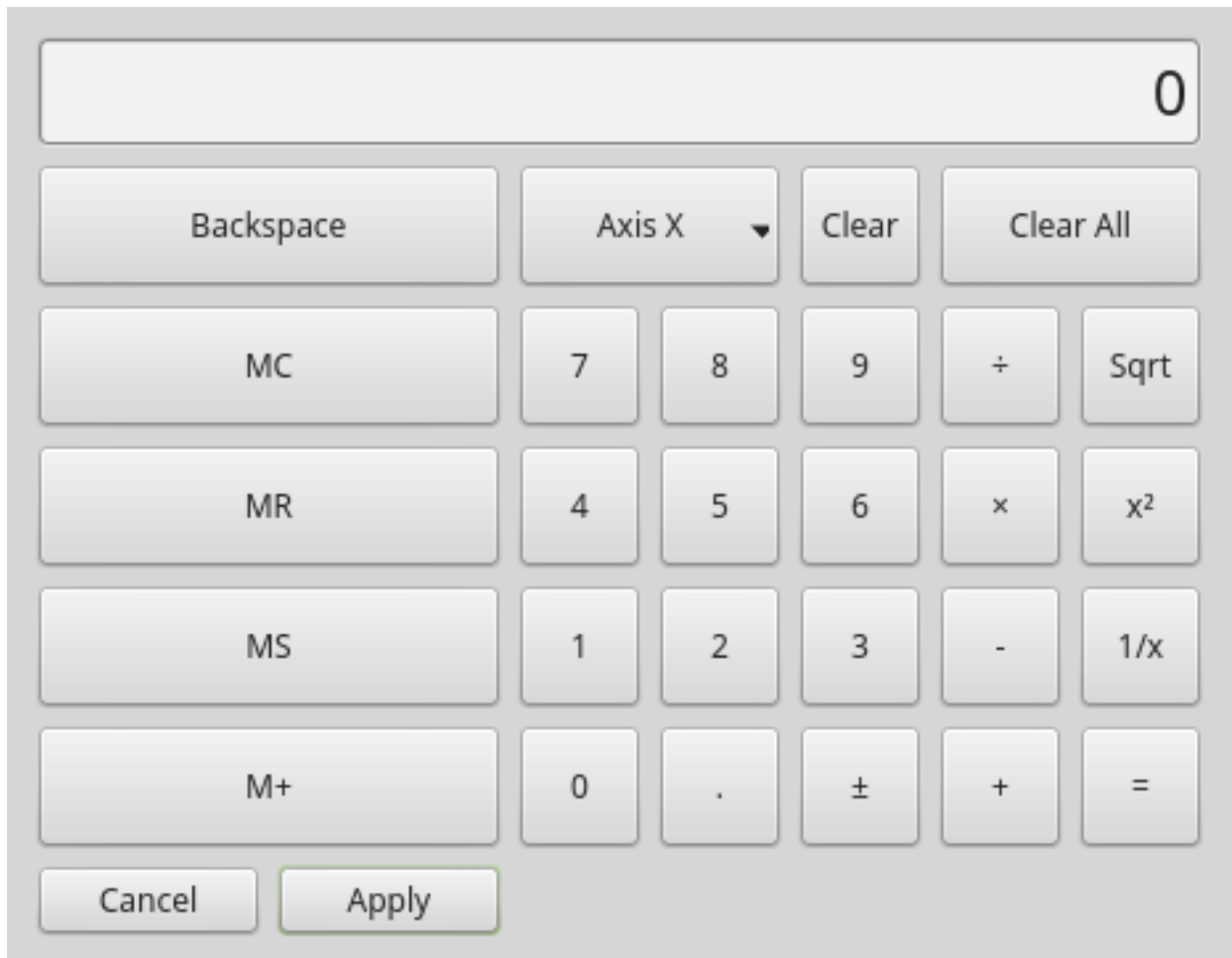


Figure 10.94: Calculator

This is a dialog to display a calculator for numeric entry, such as origin offset. It returns the entry via STATUS messages using a python DICT. The DICT contains at minimum, the name of the dialog requested and an id code. When using STATUS's *request-dialog* function, the default launch name is *CALCULATOR*. It is based on PyQt's QDialog.

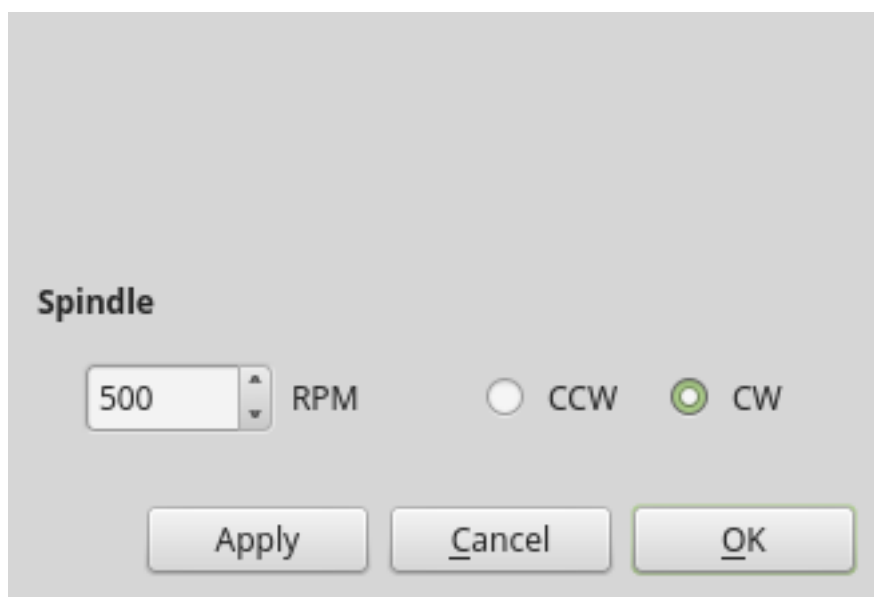


Figure 10.95: Run-from-line Dialog

Dialog to preset spindle settings before running a program from a specific line.

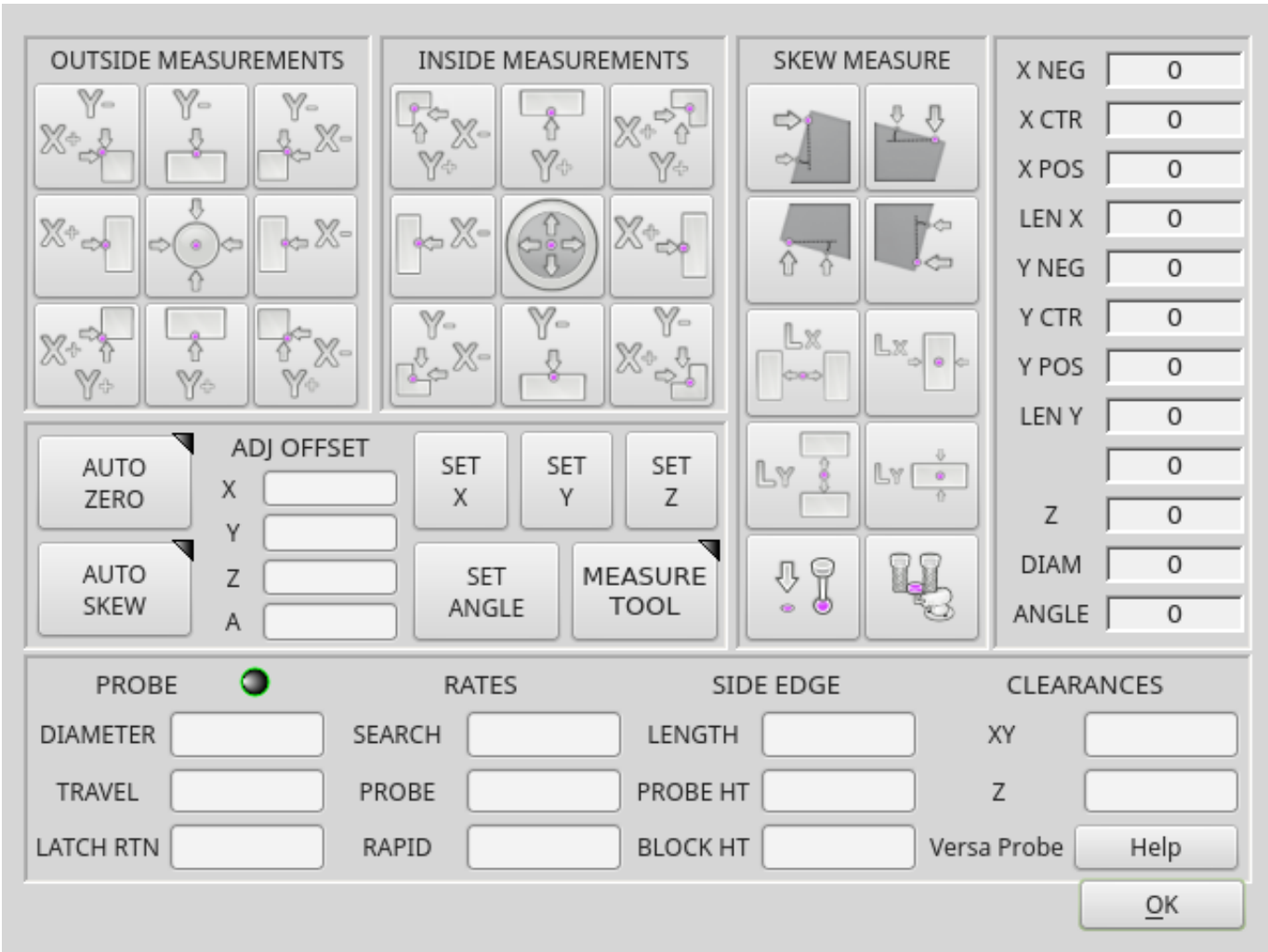


Figure 10.96: Versa Probe Dialog

This is a dialog to display A probing screen based on Versa Probe. It is based on PyQt’s QDialog

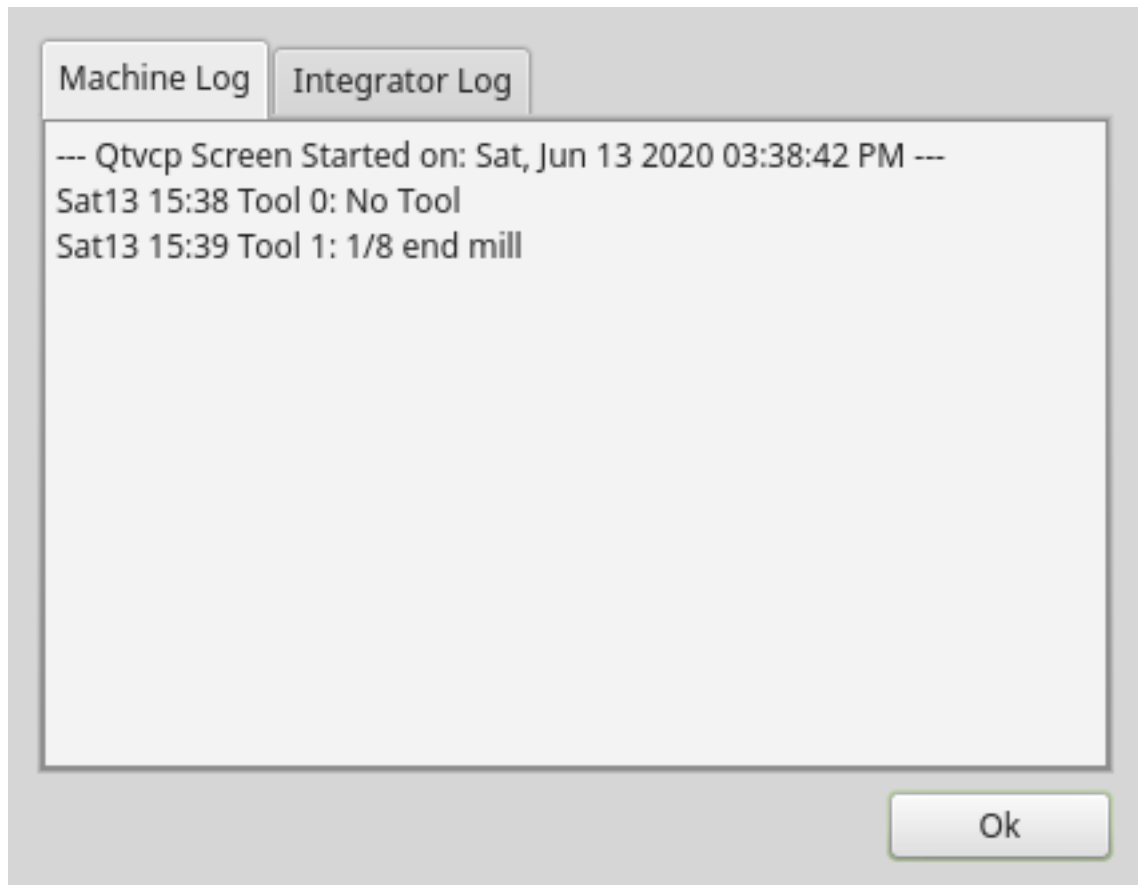


Figure 10.97: Machine Log Dialog

This is a dialog to display the user machine log and qtvcp's debugging log. It is based on PyQt's QDialog

10.4.7.5 Other

Other available widgets

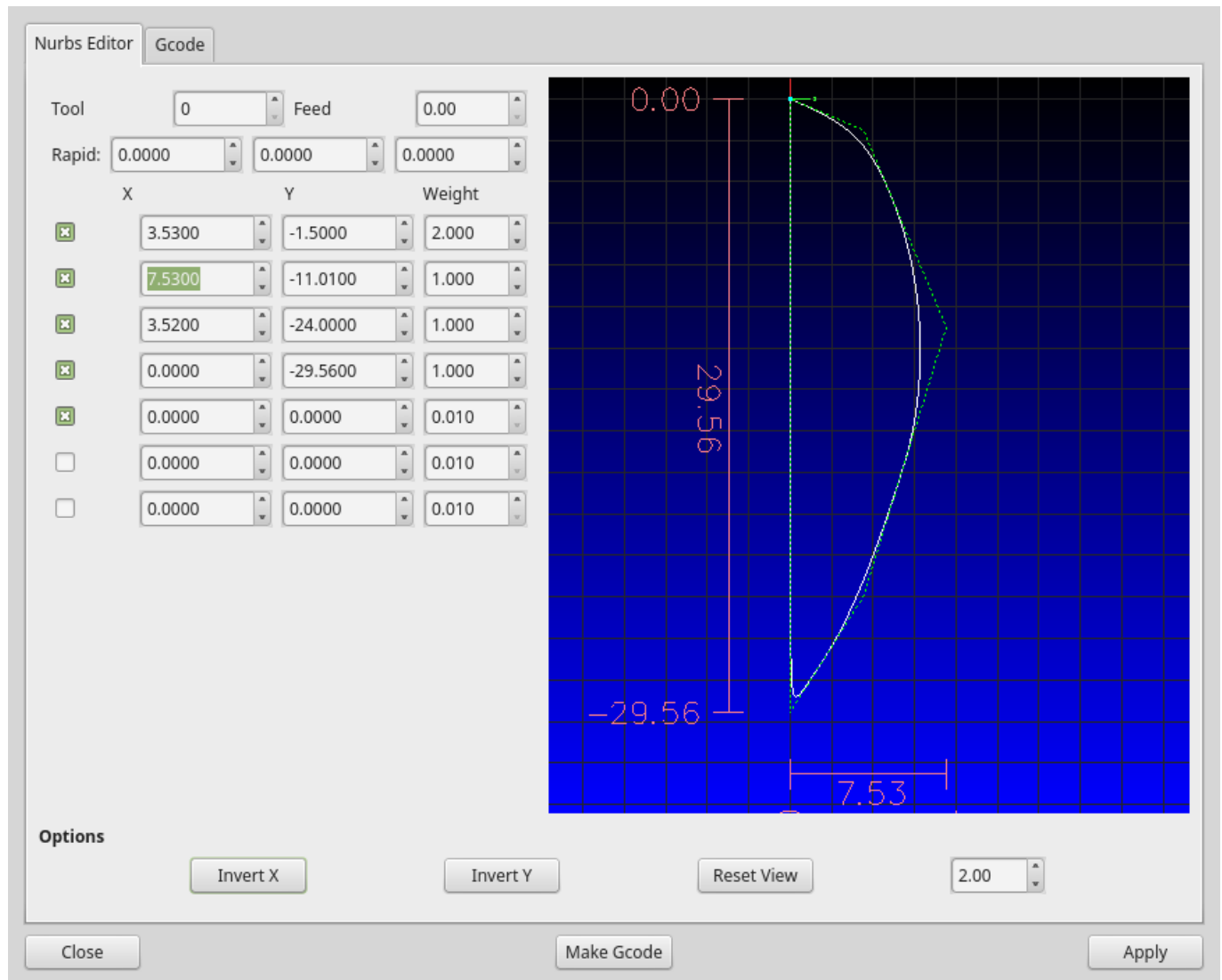


Figure 10.98: Nurbs Editor

The Nurbs editor allows you to manipulate a nurbs based geometry on screen and then convert this to G-code. you can edit the G-code on screen and then send it to LinuxCNC.

It is the base class for the HALPad widget. This widget looks and acts like a 5 button D-pad, with an LED like indicators in a ring. You can put text or icons in each of the button positions. You can connect to output signals when the buttons are pressed. There are also input slots to change the color of the indicator(s).

There are enumerated constants used to reference indicator positions. They are used in the Designer editor's property editor or used if using python code.

```
NONE
LEFT
RIGHT
CENTER
TOP
BOTTOM
LEFTRIGHT
TOPBOTTOM
```


For python handler code, you use the widget Designer name plus the reference constant.

```
self.w.joypadname.set_highlight(self.w.joypadname.LEFT)
```

As coded they issue signals for the button pressed or released. On signal outputs a string code for the button, one signal outputs a bool value.

```
def _pressedOutput(self, btncode):
    self.joy_btn_pressed.emit(btncode)
    self['joy_{}_pressed'.format(btncode.lower())].emit(True)

def _releasedOutput(self, btncode):
    self.joy_btn_released.emit(btncode)
    self['joy_{}_pressed'.format(btncode.lower())].emit(False)
```

- *reset_highlight()*:
Clears the highlight indicator.
- *set_highlight(button, state=True)*:
Set the highlight indicator in position *button* to state *state* You can use strings letters (LRCTBXA) or position ENUMS for the button argument.
- *set_button_icon(button, pixmap)*:
Sets the button's icon pixmap.
- *set_button_text(button, text)*:
Sets the button's icon text.
- *set_tooltip(button, text)*:
Sets the buttons popup tooltip descriptive text.
- *setLight(state)*: Sets the highlight indicator to the true color or false color. The *set_highlight()* function must be used prior to set the indicator to use.

These signals will be sent when buttons are pressed. They can be connected to in the Designer editor or python code. The first two output a string the indicates the button pressed.

```
joy_btn_pressed = QtCore.pyqtSignal(str)
joy_btn_released = QtCore.pyqtSignal(str)
joy_l_pressed = QtCore.pyqtSignal(bool)
joy_l_released = QtCore.pyqtSignal(bool)
joy_r_pressed = QtCore.pyqtSignal(bool)
joy_r_released = QtCore.pyqtSignal(bool)
joy_c_pressed = QtCore.pyqtSignal(bool)
joy_c_released = QtCore.pyqtSignal(bool)
joy_t_pressed = QtCore.pyqtSignal(bool)
joy_t_released = QtCore.pyqtSignal(bool)
joy_b_pressed = QtCore.pyqtSignal(bool)
joy_b_released = QtCore.pyqtSignal(bool)
```

Slots can be connected to in the Designer editor or python code.

```
set_colorStateTrue()
set_colorStateFalse()
set_colorState(bool)

set_true_color(str)
set_true_color(qcolor)

set_false_color(str)
set_false_color(qcolor)
```

These can be set in stylesheets or python code to change it's properties.

- *highlightPosition*:
Set the indicator position.
- *setColorState*:
Select the color state of the indicator.
- *left_image_path*:
- *right_image_path*:
- *center_image_path*:
- *top_image_path*:
- *bottom_image_path*:
A file path or resource path to an image to display in the described button location. If the reset button is pressed in the Designer editor property, the image will not be displayed (allowing optionally text).
- *left_text*:
- *right_text*:
- *center_text*:
- *top_text*:
- *bottom_text*:
A text string to be displayed in the described button location. If left blank an image can be designated to be displayed.
- *true_color*:
- *false_color*:
Color selection for the center LED ring to be displayed when the *BASENAME.light.center* HAL pin is True or False.
- *text_color*:
Color selection for the button text.
- *button_font*:
Font selection for the button text.

The above properties could be set in styles sheets. You would usually use the designer widget name with # to set individual widget properties, other wise you the class name *JoyPad* to set all JoyPad widgets the same.

```
#joypadname{
qproperty-true_color: #000;
qproperty-false_color: #444;
}
```

```
self.w.joypadname.setProperty('true_color','green')
self.w.joypadname.setProperty('false_color','red')
```

10.4.7.6 Import only Widgets

These widgets are usually the base class widget for other QtVCP widgets. They are not available directly from the Designer editor but could be imported and manually inserted. They could also be subclassed to make a similar widget with new features.

10.4.8 Qtvcp Libraries modules

Libraries are prebuilt python modules that give added features to Qtvcp. In this way you can select what features you want - yet don't have to build common ones yourself.

10.4.8.1 STATUS

STATUS is a library that sends GObject messages based on LinuxCNC's current state. It is an extension of gladevcp's GStat object. [GStats](#)

It also has some functions to give status on such things as internal jog rate. You connect a function call to the STATUS message you are interested in. Qtvcp will call this function when the message is sent from STATUS.

To import this modules add this python code to your import section:

```
#####
# **** IMPORT SECTION **** #
#####

from qtvcp.core import Status
```

To instantiate the module so you can use it add this python code to your instantiate section:

```
STATUS = Status()
```

To connect to messages use GObject syntax. For example to catch machine on and off messages,

Note

This example shows the two common ways of connecting signals.

Notice one uses lambda. lambda is used to strip off or manipulate arguments from the status message before calling the function.

You can see the difference in the called function signature. the one that uses lambda does not accept the status object - lambda did not pass it to the function.

place these commands into the INITIALIZE section:

```
STATUS.connect('state-on', self.on_state_on)
STATUS.connect('state-off', lambda: w, self.on_state_off())
```

These would call functions that looks like these:

```
def on_state_on(self, status_object):
    print('Linuxcnc machine is on')
def on_state_off(self):
    print('Linuxcnc machine is off')
```

In this example code when LinuxCNC is in *machine on*-state the function self.on_state_on will be called.

When LinuxCNC is in *machine off*-state the function self.on_state_off will be called.

10.4.8.2 INFO

INFO is a library to collect and filters data from the INI file.

The available data and defaults:

```

LINUXCNC_IS_RUNNING
LINUXCNC_VERSION
INIPATH
INI = linuxcnc.ini(INIPATH)
MDI_HISTORY_PATH = '~/.axis_mdi_history'
QTVCP_LOG_HISTORY_PATH = '~/.qtvcp.log'
MACHINE_LOG_HISTORY_PATH = '~/.machine_log_history'
PREFERENCE_PATH = '~/.Preferences'
SUB_PATH = None
SUB_PATH_LIST = []
self.MACRO_PATH = None
MACRO_PATH_LIST = []
INI_MACROS = self.INI.findall("DISPLAY", "MACRO")

IMAGE_PATH = IMAGEDIR
LIB_PATH = os.path.join(HOME, "share", "qtvcp")

PROGRAM_FILTERS = None
PARAMETER_FILE = None
MACHINE_IS_LATHE = False
MACHINE_IS_METRIC = False
MACHINE_UNIT_CONVERSION = 1
MACHINE_UNIT_CONVERSION_9 = [1]*9
TRAJ_COORDINATES =
JOINT_COUNT = int(self.INI.find("KINS", "JOINTS") or 0)
AVAILABLE_AXES = ['X', 'Y', 'Z']
AVAILABLE_JOINTS = [0, 1, 2]
GET_NAME_FROM_JOINT = {0: 'X', 1: 'Y', 2: 'Z'}
GET_JOG_FROM_NAME = {'X': 0, 'Y': 1, 'Z': 2}
NO_HOME_REQUIRED = False
HOME_ALL_FLAG
JOINT_TYPE = self.INI.find(section, "TYPE") or "LINEAR"
JOINT_SEQUENCE_LIST
JOINT_SYNC_LIST

JOG_INCREMENTS = None
ANGULAR_INCREMENTS = None
GRID_INCREMENTS

DEFAULT_LINEAR_JOG_VEL = 15 units per minute
MIN_LINEAR_JOG_VEL = 60 units per minute
MAX_LINEAR_JOG_VEL = 300 units per minute

DEFAULT_ANGULAR_JOG_VEL =
MIN_ANGULAR_JOG_VEL =
MAX_ANGULAR_JOG_VEL =

MAX_FEED_OVERRIDE =
MAX_TRAJ_VELOCITY =

AVAILABLE_SPINDLES = int(self.INI.find("TRAJ", "SPINDLES") or 1)
DEFAULT_SPINDLE_0_SPEED = 200
MAX_SPINDLE_0_SPEED = 2500
MAX_SPINDLE_0_OVERRIDE = 100
MIN_SPINDLE_0_OVERRIDE = 50

```

```

MAX_FEED_OVERRIDE = 1.5
MAX_TRAJ_VELOCITY

# user message dialog info
USRMESS_BOLDTEXT = self.INI.findall("DISPLAY", "MESSAGE_BOLDTEXT")
USRMESS_TEXT = self.INI.findall("DISPLAY", "MESSAGE_TEXT")
USRMESS_TYPE = self.INI.findall("DISPLAY", "MESSAGE_TYPE")
USRMESS_PINNAME = self.INI.findall("DISPLAY", "MESSAGE_PINNAME")
USRMESS_DETAILS = self.INI.findall("DISPLAY", "MESSAGE_DETAILS")
USRMESS_ICON = self.INI.findall("DISPLAY", "MESSAGE_ICON")
ZIPPED_USRMESS =

self.GLADEVCP = (self.INI.find("DISPLAY", "GLADEVCP")) or None

# embedded program info
TAB_NAMES = (self.INI.findall("DISPLAY", "EMBED_TAB_NAME")) or None
TAB_LOCATION = (self.INI.findall("DISPLAY", "EMBED_TAB_LOCATION")) or []
TAB_CMD = (self.INI.findall("DISPLAY", "EMBED_TAB_COMMAND")) or None
ZIPPED_TABS =

MDI_COMMAND_LIST =      (heading: [MDI_COMMAND_LIST], title: MDI_COMMAND")
TOOL_FILE_PATH =        (heading: [EMCIO], title:TOOL_TABLE)
POSTGUI_HALFILE_PATH =  (heading: [HAL], title: POSTGUI_HALFILE)

```

There are some *helper functions* - mostly used for widget support

```

get_error_safe_setting(self, heading, detail, default=None)
convert_metric_to_machine(data)
convert_imperial_to_machine(data)
convert_9_metric_to_machine(data)
convert_9_imperial_to_machine(data)
convert_units(data)
convert_units_9(data)
get_filter_program(fname)
get_qt_filter_extensions()

```

To import this modules add this python code to your import section:

```

#####
# **** IMPORT SECTION **** #
#####

from qtvcp.core import Info

```

To instantiate the module so you can use it in a handler file add this python code to your instantiate section:

```

#####
# **** INSTANTIATE LIBRARIES SECTION **** #
#####

INFO = Info()

```

To access INFO data use this general syntax:

```

home_state = INFO.NO_HOME_REQUIRED
if INFO.MACHINE_IS_METRIC is True:
    print('Metric based')

```

10.4.8.3 Action

This library is used to command LinuxCNC's motion controller.
It tries to hide incidental details and add convenience methods for developers.

To import this modules add this python code to your import section:

```
#####
# **** IMPORT SECTION **** #
#####

from qtvcp.core import Action
```

To instantiate the module so you can use it add this python code to your instantiate section:

```
#####
# **** INSTANTIATE LIBRARIES SECTION **** #
#####

ACTION = Action()
```

To access Action commands use general syntax such as these:

```
ACTION.SET_ESTOP_STATE(state)
ACTION.SET_MACHINE_STATE(state)

ACTION.SET_MACHINE_HOMING(joint)
ACTION.SET_MACHINE_UNHOMED(joint)

ACTION.SET_LIMITS_OVERRIDE()

ACTION.SET_MDI_MODE()
ACTION.SET_MANUAL_MODE()
ACTION.SET_AUTO_MODE()

ACTION.SET_LIMITS_OVERRIDE()

ACTION.CALL_MDI(code)
ACTION.CALL_MDI_WAIT(code)
ACTION.CALL_INI_MDI(number)

ACTION.CALL_OWORD()

ACTION.OPEN_PROGRAM(filename)
ACTION.SAVE_PROGRAM(text_source, fname):

ACTION.SET_AXIS_ORIGIN(axis,value)
ACTION.SET_TOOL_OFFSET(axis,value,fixture = False)

ACTION.RUN()
ACTION.ABORT()
ACTION.PAUSE()

ACTION.SET_MAX_VELOCITY_RATE(rate)
ACTION.SET_RAPID_RATE(rate)
ACTION.SET_FEED_RATE(rate)
ACTION.SET_SPINDLE_RATE(rate)

ACTION.SET_JOG_RATE(rate)
ACTION.SET_JOG_INCR(incr)
```

```
ACTION.SET_JOG_RATE_ANGULAR(rate)
ACTION.SET_JOG_INCR_ANGULAR(incr, text)

ACTION.SET_SPINDLE_ROTATION(direction = 1, rpm = 100, number = 0)
ACTION.SET_SPINDLE_FASTER(number = 0)
ACTION.SET_SPINDLE_SLOWER(number = 0)
ACTION.SET_SPINDLE_STOP(number = 0)

ACTION.SET_USER_SYSTEM(system)

ACTION.ZERO_G92_OFFSET()
ACTION.ZERO_ROTATIONAL_OFFSET()
ACTION.ZERO_G5X_OFFSET(num)

ACTION.RECORD_CURRENT_MODE()
ACTION.RESTORE_RECORDED_MODE()

ACTION.SET_SELECTED_AXIS(jointnum)

ACTION.DO_JOG(jointnum, direction)
ACTION.JOG(jointnum, direction, rate, distance=0)

ACTION.TOGGLE_FLOOD()
ACTION.SET_FLOOD_ON()
ACTION.SET_FLOOD_OFF()

ACTION.TOGGLE_MIST()
ACTION.SET_MIST_ON()
ACTION.SET_MIST_OFF()

ACTION.RELOAD_TOOLTABLE()
ACTION.UPDATE_VAR_FILE()

ACTION.TOGGLE_OPTIONAL_STOP()
ACTION.SET_OPTIONAL_STOP_ON()
ACTION.SET_OPTIONAL_STOP_OFF()

ACTION.TOGGLE_BLOCK_DELETE()
ACTION.SET_BLOCK_DELETE_ON()
ACTION.SET_BLOCK_DELETE_OFF()

ACTION.RELOAD_DISPLAY()
ACTION.SET_GRAPHICS_VIEW(view)

ACTION.UPDATE_MACHINE_LOG(text, option=None):

ACTION.CALL_DIALOG(command):

ACTION.HIDE_POINTER(state):

ACTION.PLAY_SOUND(path):
ACTION.PLAY_ERROR():
ACTION.PLAY_DONE():
ACTION.PLAY_READY():
ACTION.PLAY_ATTENTION():
ACTION.PLAY_LOGIN():
ACTION.PLAY_LOGOUT():
ACTION.SPEAK(speech):

ACTION.BEEP():
ACTION.BEEP_RING():
ACTION.BEEP_START():
```

```
ACTION.SET_DISPLAY_MESSAGE(string)
ACTION.SET_ERROR_MESSAGE(string)
```

There are some *helper functions* - mostly used for this library's support

```
get_jog_info (num)
jnum_check(num)
ensure_mode(modes)
open_filter_program(filename, filter)
```

10.4.8.4 TOOL

This library handles tool offset file changes.
Linuxcnc doesn't handle third party manipulation of the tool file well.

This will return a Python list of information on the requested tool number.

This return a single python list of python lists of tool information.
This is a raw list formed from the system tool file.

This will return a python tuple of two Python lists of Python lists of tool information.
[0] will be real tools information
[1] will be wear tool information (tool numbers will be over 10000; Fanuc style tool wear)

This will by default, add a black tool entry with tool number -99.
You can preload the *newtool* array with tool information.

Delete the numbered tool

This will parse the toolarray and save it to the file specified in the INI as the tool path.
This tool array must contain all the available tools information.
This array is expected to use the LinuxCNC raw tool array - (doesn't have tool wear entries)
it will return True if there was an error.

This function converts a LinuxCNC raw tool array to a qtvcp tool wear array.
QtVCP's array includes entries for X and Z axis tool wear.
inuxcnc covers tool wear by adding tool wear information into tool entries above 10000.
This also requires remap code to add the wear offsets at tool change time.

This function converts qtvcp's tool array into a LinuxCNC raw tool array.
QtVCP's array includes entries for X and Z axis tool wear.
LinuxCNC covers tool wear by adding tool wear information into tool entries above 10000.
This also requires remap code to add the wear offsets t tool change time.

10.4.8.5 Path

This module give reference to important file paths.
To import this modules add this python code to your import section:


```
#####
# **** IMPORT SECTION **** #
#####

from qtvcp,core import Path
```

To instantiate the module so you can use it add this python code to your instantiated section:

```
#####
# **** INSTANTIATE LIBRARIES SECTION **** #
#####

PATH = Path()

PATH.PREFS_FILENAME # The preference file path
PATH.WORKINGDIR      # The directory qtvcp was launched from
PATH.IS_SCREEN       # Is this a screen or a VCP?
PATH.CONFIGPATH      # launched configuration folder
PATH.RIPCONFIGDIR    # The Run-in-place config folder for QtVCP screens
PATH.BASDIR          # Base folder for LinuxCNC
PATH.BASENAME        # The Designer files name (no ending)
PATH.IMAGEDIR        # The QtVCP image folder
PATH.SCREENDIR       # The QtVCP builtin Screen folder
PATH.PANELDIR        # The QtVCP builtin VCP folder
PATH.HANDLER         # Handler file Path
PATH.XML             # QtVCP UI file path
PATH.QSS             # QtVCP Qss file path
PATH.PYDIR           # Linuxcnc's python library
PATH.LIBDIR          # The QtVCP library folder
PATH.WIDGET          # The QtVCP widget folder
PATH.PLUGIN          # The QtVCP widget plugin folder

# Not currently used:
PATH.LOCALEDIR       # Locale translation folder
PATH.DOMAIN          # Translation domain
```

10.4.8.6 VCPWindow

This module give reference to the Main window and widgets.
Typically this would be used for a library (eg, toolbar library uses it.) as
the widgets get a reference to the main window from the `_hal_init()` function.

To import this modules add this python code:

```
#####
# **** IMPORT SECTION **** #
#####

from qtvcp.qt_makegui import VCPWindow
```

To instantiate the module so you can use it add this python code:

```
#####
# **** INSTANTIATE LIBRARIES SECTION **** #
#####

WIDGETS = VCPWindow()
```

10.4.8.7 aux_program_loader

This module allows an easy way to load auxiliary programs LinuxCNC often uses. To import this modules add this python code to your import section:

```
#####
# **** IMPORT SECTION **** #
#####

from qtvcp.lib.aux_program_loader import Aux_program_loader
```

To instantiate the module so you can use it add this python code to your instantiated section:

```
#####
# **** INSTANTIATE LIBRARIES SECTION **** #
#####

AUX_PRGM = Aux_program_loader()
```

Halmeter is used to display one HAL pin data. load a halmeter with:

```
AUX_PRGM.load_halmeter()
```

```
AUX_PRGM.load_ladder()
```

```
AUX_PRGM.load_status()
```

```
AUX_PRGM.load_halshow()
```

```
AUX_PRGM.load_halscope()
```

```
AUX_PRGM.load_tooledit(TOOLEFILE_PATH)
```

```
AUX_PRGM.load_calibration()
```

```
AUX_PRGM.keyboard_onboard(ARGS)
```

10.4.8.8 Keybindings

This module is used to allow each keypress to control a behavior such as jogging. It's used inside the handler file to facilitate keyboard jogging etc.

To import this modules add this python code to your import section:

```
#####
# **** IMPORT SECTION **** #
#####

from qtvcp.lib.keybindings import Keylookup
```

To instantiate the module so you can use it add this python code to your instantiate section:

```
#####
# **** INSTANTIATE LIBRARIES SECTION **** #
#####

KEYBIND = Keylookup()
```

Note keybindings require code under the *processed_key_event* function to call *KEYBIND.call()* - most handler file already have this code.
In the handler file, under the *initialed* function use this general syntax:

```
KEYBINDING.add_call("DEFINED_KEY","FUNCTION TO CALL", USER DATA)
```

Here we add a keybinding for F10, 11 and 12

```
#####
# Special Functions called from QTVCP
#####

# at this point:
# the widgets are instantiated.
# the HAL pins are built but HAL is not set ready
def initialized__(self):
    KEYBIND.add_call('Key_F10','on_keycall_F10',None)
    KEYBIND.add_call('Key_F11','on_keycall_override',10)
    KEYBIND.add_call('Key_F12','on_keycall_override',20)
```

And then need to add functions that get called:
In the handler file, under the *KEY BINDING CALLS* section add this:

```
#####
# KEY BINDING CALLS #
#####

def on_keycall_F12(self,event,state,shift,cntrl,value):
    if state:
        print('F12 pressed')

def on_keycall_override(self,event,state,shift,cntrl,value):
    if state:
        print('value = {}'.format(value))
```

10.4.8.9 Messages

This modules is used to display pop up dialog messages on the screen.
These are defined in the INI file and controlled by HAL pins.

Boldtext is generally a title.

text is below that and usually longer.

Detail is hidden unless clicked on.

pinname is the basename of the HAL pins.

type specifies whether its a yes/no, ok, or status message.

Status messages will be shown in the status bar and the notify dialog.

it requires no user intervention.

ok messages require the user to click ok to close the dialog.

ok messages have one HAL pin to launch the dialog and one to signify it's waiting for response.

yes/no messages require the user to select yes or no buttons to close the dialog.

yes/no messages have three hal pins - one to show the dialog, one for waiting, and one for the answer.

By default it will send STATUS messages for focus_overlay anf alert sound.

These will silently fail if those options are not set up.

Here is a sample INI code. It would be under the [DISPLAY] heading.

```
# This just shows in the status bar and desktop notify popup.
MESSAGE_BOLDTEXT = NONE
MESSAGE_TEXT = This is a statusbar test
MESSAGE_DETAILS = STATUS DETAILS
MESSAGE_TYPE = status
MESSAGE_PINNAME = statustest

# This will pop up a dialog that asks a yes no question
MESSAGE_BOLDTEXT = NONE
MESSAGE_TEXT = This is a yes no dialog test
MESSAGE_DETAILS = Y/N DETAILS
MESSAGE_TYPE = yesnodialog
MESSAGE_PINNAME = yndialogtest

# This pops up a dialog that requires an ok response and it shows in the status bar and
# the desktop notify popup.
MESSAGE_BOLDTEXT = This is the short text
MESSAGE_TEXT = This is the longer text of the both type test. It can be longer then the ↔
               status bar text
MESSAGE_DETAILS = BOTH DETAILS
MESSAGE_TYPE = okdialog status
MESSAGE_PINNAME = bothtest
```

The screenoptions widget can automatically set up the message system.

10.4.8.10 notify

This module is used to send messages that are integrated into the desktop.
it uses the pynotify library.

Ubuntu/Mint does not follow the standard - you can't set how long the message stays up for.
I suggest fixing this with a PPA off the net

https://launchpad.net/~leolik/+archive/leolik?field.series_filter=lucid

you can set the title, message, icon, and timeout.

Notify keeps a list of all the alarm messages since starting in self.alarmpage.

If you click *show all messages* in the notify popup, it will print then to the terminal.

The screenoptions widget can automatically set up the notify system.
Typically STATUS messages are used to sent notify messages.

10.4.8.11 preferences

This module allows one to load and save preference data permanently to storage media.

The screenoptions widget can automatically set up the preference system.

qtvcp searches for the screenoption widget first and if found calls `_pref_init()`.

This will create the preference object and return it to qtvcp to pass to all the widgets and add it to the window object attributes. In this case the preferences object would be accessible from the handler file's `initialized_` method as `self.w.PREFS_`

Also all widgets can have access to a specific preference file at initialization time.

The screenoptions widget can automatically set up the preference file.

10.4.8.12 audio player

This module allows playing sounds using gstreamer, beep and espeak
 It can play sounds/music files using *gstreamer* (non blocking)
 It can play sounds using the *beep* library (currently blocks while beeping)
 It can speak using the *espeak* library (non blocks while speaking)
 There are default alert sounds using Mint or freedesktop default sounds.
 You can play arbitrary sounds or even songs by specifying the path.
 STATUS has messages to control this module.
 The screenoptions widget can automatically set up the audio system.
 Or to import this modules manually add this python code to your import section:

```
#####
# **** IMPORT SECTION **** #
#####

from qtvcp.lib.audio_player import Player
```

To instantiate the module so you can use it add this python code to your instantiated section:
 _register_messages() function connects the audio player to the STATUS library for sounds can be played with the STATUS message system.

```
#####
# **** INSTANTIATE LIBRARIES SECTION **** #
#####

SOUND = Player()
SOUND._register_messages()
```

To play sounds using STATUS messages use these general syntax:

```
STATUS.emit('play-alert', 'LOGOUT')
STATUS.emit('play-alert', 'BEEP')
STATUS.emit('play-alert', 'SPEAK This is a test screen for Qt V C P')
STATUS.emit('play-sound', 'PATH TO SOUND')
```

There are default alerts to choose from:

```
ERROR
READY
ATTENTION
RING
DONE
LOGIN
LOGOUT
```

There are three beeps:

```
BEEP_RING
BEEP_START
BEEP
```

10.4.8.13 virtual keyboard

This library allows you to use STATUS messages to launch a virtual keyboard. It uses Onboard or Matchbox library for the keyboard.

10.4.8.14 TOOLBAR ACTIONS

This library supplies prebuilt submenus and actions for toolbar menus and toolbar buttons. Toolbuttons, menu and toolbar menus are built in designer and assigned actions/submenus in the handler file.

Actions:

```
estop
power
load
reload
gcode_properties
run
pause
abort
block_delete
optional_stop
touchoffworkplace
touchofffixture
runfromline
load_calibration
load_halmeter
load_halshow
load_status
load_halscope
about
zoom_in
zoom_out
view_x
view_y
view_y2
view_z
view_z2
view_p
view_clear
show_offsets
quit
system_shutdown
tooloffsetdialog
originoffsetdialog
calculatordialog
alphamode
inhibit_selection
show_dimensions
```

Submenus:

```
recent_submenu
home_submenu
unhome_submenu
zero_systems_submenu
grid_size_submenu
```

Here is the typical code to add to the relevant handler file sections.

```
#####
# **** IMPORT SECTION **** #
#####

from qtvcp.lib.toolbar_actions import ToolBarActions

#####
# **** instantiate libraries section **** #
#####

TOOLBAR = ToolBarActions()

#####
# Special Functions called from QTVCP
#####

# at this point:
# the widgets are instantiated.
# the HAL pins are built but HAL is not set ready
def initialized__(self):
    TOOLBAR.configure_submenu(self.w.menuHoming, 'home_submenu')
    TOOLBAR.configure_action(self.w.actionEstop, 'estop')
    TOOLBAR.configure_action(self.w.actionQuit, 'quit', lambda d:self.w.close())
    TOOLBAR.configure_action(self.w.actionEdit, 'edit', self.edit)
    # Add a custom function
    TOOLBAR.configure_action(self.w.actionMyFunction, 'my_Function', self.my_function)
```

In the *GENERAL FUNCTIONS SECTION* ADD:

```
#####
# general functions #
#####

def my_function(self, widget, state):
    print('My function State = {}'.format(state))
```

10.4.8.15 QT Vismach Machine Graphics library

Qt vismach is a set of Python functions that can be used to create and animate models of machines. Vismach displays the model in a 3D viewport and the model parts are animated as the values of associated

HAL pins change.

This is the Qt based version of the library, there is also a tkinter version available in linuxcnc. the Qt version allows embedding the simulation in other screens.

There are included sample panels in QTVcp for a 3 axis XYZ mill, a 5 axis gantry a 3 axis mill with an A axis/spindle and a scara mill.

These samples, if loaded after a running linuxcnc configuration (including non-QTVcp based screens), most will react to machine movement. Some require HAL pins to be connected for movement

from a terminal (pick one):

```
qtvcp vismach_mill_xyz
qtvcp vismach_scara
qtvcp vismach_millturn
qtvcp vismach_5axis_gantry
```

This gives basic building blocks of a simulated machine.

A collection is an object of individual machine parts. This holds a hierarchical list of primitive shapes or STL objects that operations can be applied to.

This object will perform an OpenGL translation calculation on a Collection object. Translation refers to moving an object to a different position on screen.

This object will perform an OpenGL scale function on a collection object.

This object will perform an OpenGL translation calculation on a Collection object, offset by the HAL pin value. Translation refers to moving an object to a different position on screen.

You can either read a pin from a component owned by the vismach object or, if the component argument is set to None, read a HAL system pin directly.

This object will perform an OpenGL rotation calculation on a Collection object,

This object will perform an OpenGL rotation calculation on a Collection object, offset by the HAL pin value.

You can either read a pin from a component owned by the vismach object or, if the component argument is set to None, read a HAL system pin directly.

This object will build a CylinderZ object that will change size and length based on loaded tool dimension (from the tool table)

It reads the HAL pins: *halui.tool.diameter* and *motion.tooloffset.z*

from mill_xyz sample

```
toolshape = CylinderZ(0)
toolshape = Color([1, .5, .5, .5], [toolshape])
tool = Collection([
    Translate([HalTranslate([tooltip], None, "motion.tooloffset.z", 0, 0, -MODEL_SCALING)], ←
        0, 0, 0),
    HalToolCylinder(toolshape)
])
```

Move and rotate an object to point from one capture()'d coordinate system to another.

Base object to hold coordinates for primitive shapes.

Builds a cylinder on the X (or Y or Z) axis by giving endpoint X (or Y, or Z) and radii coordinates.

Builds a sphere from center and radius coordinates.

Builds a triangle in the soecified plane by giving the corners Z coordinates for each side.

Build an arc by specifying:

Build a box specified by the 6 coordinates.

Specify the width in X and Y, and the height in Z the box is centered on the origin.

specify the width in X and Y, and the height in Z the box is centered in X and Y, and runs from Z=0 up (or down) to the specified Z value.

capture current transformation matrix of a collection.

note that this transforms from the current coordinate system to the viewport system, NOT to the world system

heads up display

draws a semi-transparent text box.

use HUD.strs for things that must be updated constantly,

and HUD.show("stuff") for one-shot things like error messages

Applies a color to the parts of a collection.

Loads a STL file/ STL data as a vismach part.

Here is how one might import the XYZ_mill simulation in a qtvcp panel or screen handler file.

Import the Mill library:


```
#####
# **** IMPORT SECTION **** #
#####

import mill_xyz as MILL
```

Instantiate the simulation widget and add it to the screen's main layout:

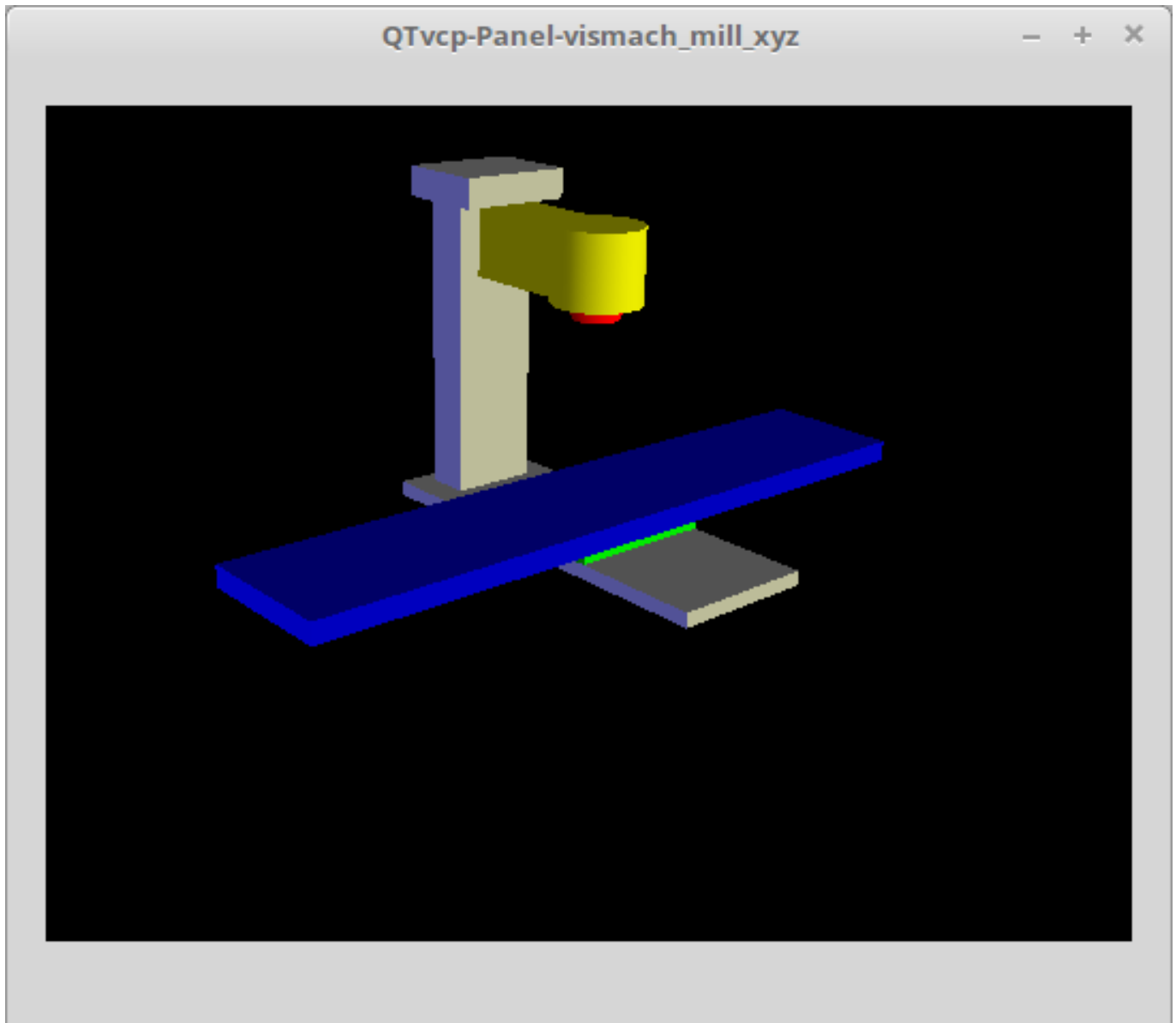
```
#####
# Special Functions called from QTVCP
#####

# at this point:
# the widgets are instantiated.
# the HAL pins are built but HAL is not set ready
def initialized__(self):
    machine = MILL.Window()
    self.w.mainLayout.addWidget(machine)
```

More information on how to build a custom machine simulation. [QtVCP Vismach](#)

10.4.9 QT Vismach

Vismach is a set of Python functions that can be used to create and animate models of machines. Vismach displays the model in a 3D viewport and the model parts are animated as the values of associated HAL pins change.



The Vismach viewport view can be manipulated as follows

- zoom by scroll wheel
- pan by middle button drag
- rotate by right-button drag
- tilt by left button drag

A Vismach model takes the form of a Python script and can use standard Python syntax. This means that there is more than one way to lay out the script, but in the examples given in this document

I will use the simplest and most basic of them.

The basic sequence in creating the Vismach model is

- Create the parts
- Define how they move
- Assemble into movement groups

10.4.9.1 Hierarchy of Machine Design

The model follows logical tree design - picture the tree, with branch and smaller branches off it if you move the larger branch, smaller branches will move with it, but if you move smaller branch larger will not.

Machine design follows that conceptual design, so for example if you move X, it can move on its own, but if you move Y, it will also move X assembly, as it is attached to Y assembly.

So for this machine, tree looks like this:

```
model
|
|---frame
|   |
|   |---base
|   |
|   |---column
|   |
|   |---top
|
|---yassembly
|   |
|   |---xassembly
|   |   |
|   |   |---xbase
|   |   |
|   |   |---work
|   |
|   |---ybase
|
|---zassembly
|   |
|   |---zframe
|   |   |
|   |   |---zbody
|   |   |
|   |   |---spindle
|   |
|   |---toolassembly
|   |   |
|   |   |---cat30
|   |   |
|   |   |---tool
|   |   |   |
|   |   |   |---tooltip
|   |   |   |
|   |   |   |---(tool cylinder function)
```

As you can see, lowest parts must exist first before it can be grouped with others into assembly. So you build upwards from lowest point in tree and assembly them together.

Same is applicable for any design of machine. Look at machine arm example and you will see that it starts with tip and adds to larger part of arm then it finally groups with base.

10.4.9.2 Start the script

It is useful for testing to include the `#!/usr/bin/env python3` to allow the file to be run as a script. The first thing to do is to import the required libraries.

```
#!/usr/bin/env python3
```

```
import hal
import math
import sys

from qtvcp.lib.qt_vismach.qt_vismach import *
```

10.4.9.3 HAL pins.

Originally the vismach library required creating a component and connecting HAL pins to control the simulation.

qt_vismach can read the system HAL system pins directly or if you wish, to use separate HAL pins that you must define in a HAL component:

```
c = hal.component("samplegui")
c.newpin("joint0", hal.HAL_FLOAT, hal.HAL_IN)
c.newpin("joint1", hal.HAL_FLOAT, hal.HAL_IN)
c.ready()
```

10.4.9.4 Creating Parts

It is probably easiest to create geometry in a CAD package and import into the model script with the AsciiSTL() or AsciiOBJ() functions.

Both functions can take one of two named arguments, either a filename or raw data.

- `part = AsciiSTL(filename="path/to/file.stl)` + `part = AsciiSTL(data="solid part1 facet normal")` + `part = AsciiOBJ(filename="path/to/file.obj)` + `part = AsciiOBJ(data="v 0.123 0.234 0.345 1.0 ...")`

The parts will be created in the Vismach space in the same locations as they occupy in the STL or OBJ space.

This means that it may be possible to assemble the model in the CAD package.

Alternatively parts can be created inside the model script from a range of shape primitives.

Many shapes are created at the origin and need to be moved to the required location after creation.

- `cylinder = CylinderX(x1, r1, x2, r2)` + `cylinder = CylinderY(y1, r1, y2, r2)` + `cylinder = CylinderZ(z1, r1, z2, r2)`
Creates a (optionally tapered) cylinder on the given axis with the given radii at the given points on the axis.
- `sphere = Sphere(x, y, z, r)`
Creates a sphere of radius r at (x,y,z)
- `triangle = TriangleXY(x1, y1, x2, y2, x3, y3, z1, z2)` + `triangle = TriangleXZ(x1, z1, x2, z2, x3, z3, y1, y2)` + `triangle = TriangleYZ(y1, z1, y2, z2, y3, z3, x1, x2)`
Creates a triangular plate between planes defined by the last two values parallel to the specified plane, with vertices given by the three coordinate pairs.
- `arc = ArcX(x1, x2, r1, r2, a1, a2)`
Create an arc shape.
- `box = Box(x1, y1, z1, x2, y2, z2)`
Creates a rectangular prism with opposite corners at the specified positions and edges parallel to the XYZ axes.
- `box = BoxCentered(xw, yw, zw)`
Creates an xw by yw by zw box centred on the origin.

- `box = BoxCenteredXY(xw, yw, z)`
Creates a box of width `xw` / `yw` and height `z`.

Composite parts may be created by assembling these primitives either at creation time or subsequently:

```
part1 = Collection([Sphere(100,100,100,50), CylinderX(100,40,150,30)])
part2 = Box(50,40,75,100,75,100)
part3 = Collection([part2, TriangleXY(10,10,20,10,15,20,100,101)])
part4 = Collection([part1, part2])
```

10.4.9.5 Orientating Model Parts

Parts may need to be moved in the Vismach space to assemble the model. They may also need to be moved to create the animation as the animation rotation axis is created at the origin (but moves with the Part).

- `part1 = Translate([part1], x, y, z)`
Move `part1` the specified distances in `x`, `y` and `z`.
- `part1 = Rotate([part1], theta, x, y, z)`
Rotate the part by angle `theta` about an axis between the origin and `x`, `y`, `z`.

10.4.9.6 Animating Parts

To animate the model (controlled by the values of HAL pins) there are two functions *HalTranslate*, *HalRotate* and *HalToolCylinder*.

For parts to move inside an assembly they need to have their HAL motions defined before being assembled with the "Collection" command.

The rotation axis and translation vector move with the part as it is moved by the vismach script during model assembly, or as it moves in response to the HAL pins as the model is animated.

- `part = HalTranslate([part], None, "hal_pin_name", xs, ys, zs)`
The function arguments are:
 - first a *collection/part* which can be pre-created earlier in the script, or could be created at this point if preferred eg `part1 = HalTranslate([Box(...)], ...)`.
 - The *HAL component* is the next argument.
In QtVCP if you are reading system pins directly then the component argument is set to 'None'.
 - After that is the *name of the HAL pin* that will animate the motion.
This needs to match an existing HAL pin that describes the joint position such as:

```
"joint.2.pos-fb"
```

Otherwise the component instance would be specified and the pin name of that component would be specified. ** Then follow the *X, Y, Z scales*.

For a Cartesian machine created at 1:1 scale this would typically be 1,0,0 for a motion in the positive X direction.

However if the STL file happened to be in cm and the machine was in inches, this could be fixed at this point by using 0.3937 (1cm /2.54in) as the scale.

- `part = HalRotate([part], None, "hal_pin_name", angle_scale, x, y, z)`
This command is similar in its operation to *HalTranslate* except that it is typically necessary to move the part to the origin first to define the axis.

- The *axis of rotation* is from the origin point to the point defined by (x,y,z).
When the part is moved back away from the origin to its correct location the axis of rotation can be considered to remain "embedded" in the part.
- *Rotation angles* are in degrees, so for a rotary joint with a 0-1 scaling you would need to use an angle scale of 360.

10.4.9.7 Assembling the model.

In order for parts to move together they need to be assembled with the `Collection()` command. It is important to assemble the parts and define their motions in the correct sequence.

For example to create a moving head milling machine with a rotating spindle and an animated draw bar you would:

- Create the head main body.
- Create the spindle at the origin.
- Define the rotation.
- Move the head to the spindle or spindle to the head.
- Create the draw bar
- Define the motion of the draw bar
- Assemble the three parts into a head assembly
- Define the motion of the head assembly.

In this example the spindle rotation is indicated by rotation of a set of drive dogs:

```
#Drive dogs
dogs = Box(-6,-3,94,6,3,100)
dogs = Color([1,1,1,1],[dogs])
dogs = HalRotate([dogs],c,"spindle",360,0,0,1)
dogs = Translate([dogs],-1,49,0)

#Drawbar
draw = CylinderZ(120,3,125,3)
draw = Color([1,0,.5,1],[draw])
draw = Translate([draw],-1,49,0)
draw = HalTranslate([draw],c,"drawbar",0,0,1)

# head/spindle
head = AsciiSTL(filename="./head.stl")
head = Color([0.3,0.3,0.3,1],[head])
head = Translate([head],0,0,4)
head = Collection([head, tool, dogs, draw])
head = HalTranslate([head],c,"Z",0,0,0.1)

# base
base = AsciiSTL(filename="./base.stl")
base = Color([0.5,0.5,0.5,1],[base])
# mount head on it
base = Collection([head, base])
```

Finally a single collection of all the machine parts, floor and work (if any) needs to be created.

For a *serial machine* each new part will be added to the collection of the previous part.

For a *parallel machine* there may be several "base" parts.

Thus, for example, in `scaragui.py` link3 is added to link2, link2 to link1 and link1 to link0, so the final model is created by:

```
model = Collection([link0, floor, table])
```

Whereas a VMC model with separate parts moving on the base might have

```
model = Collection([base, saddle, head, carousel])
```

10.4.9.8 Other functions

- `part = Color([colorspec], [part])`
Sets the display color of the part.
Note that unlike the other functions the part definition comes second in this case.
The colorspec consists of the three RGB values and an opacity.
For example `[1,0,0,0.5]` for a 50% opacity red.
 - `myhud = Hud() + myhud.show("Mill_XYZ")`
Creates a heads-up display in the Vismach GUI to display such items as axis positions, titles, or messages.
 - `part = Capture()`
This sets the current position in the model.
 - `main(model, tooltip, work, size=10, hud=myhud, rotation_vectors=None, lat=0, lon=0)`
This is the command that makes it all happen, creates the display etc. if invoked directly from python3.
Usually this file is imported by QtVCP and the `window()` object is instantiated and embedded into another screen.
- *model* should be a collection that contains all the machine parts.
 - *tooltip* and *work* need to be created by `Capture()` to visualize their motion in the backplot.
See `mill_xyz.py` for an example of how to connect the tool tip to a tool and the tool to the model.
 - Either *rotation_vectors* or *latitude/longitude* can be used to set the original viewpoint and it is advisable to do as the default initial viewpoint is rather unhelpfully from immediately overhead.
 - *size* sets the extent of the volume visualized in the initial view. *hud* refers to a head-up display of axis positions.

10.4.9.9 Basic structure of a QT Vismach script.

```
# imports
import hal
from qtvcp.lib.qt_vismach.qt_vismach import *

# create HAL pins here if needed
#c = hal.component("samplegui")
#c.newpin("joint0", hal.HAL_FLOAT, hal.HAL_IN)

# create the floor, tool and work
```

```

floor = Box(-50, -50, -3, 50, 50, 0)
work = Capture()
tooltip = Capture()

# Build and assemble the model
part1 = Collection([Box(-6, -3, 94, 6, 3, 100)])
part1 = Color([1, 1, 1, 1], [part1])
part1 = HalRotate([part1], None, "joint.0.pos-fb", 360, 0, 0, 1)
part1 = Translate([dogs], -1, 49, 0)

# create a top-level model
model = Collection([base, saddle, head, carousel])

# we want to either embed into qtvcp or display directly with PyQt5
# so build a window to display the model

class Window(QWidget):
    def __init__(self):
        super(Window, self).__init__()
        self.glWidget = GLWidget()
        v = self.glWidget
        v.set_latitudelimits(-180, 180)

        world = Capture()

        # uncomment if there is a HUD
        # HUD needs to know where to draw
        #v.hud = myhud
        #v.hud.app = v

        v.model = Collection([model, world])
        size = 600
        v.distance = size * 3
        v.near = size * 0.01
        v.far = size * 10.0
        v.tool2view = tooltip
        v.world2view = world
        v.work2view = work

        mainLayout = QHBoxLayout()
        mainLayout.addWidget(self.glWidget)
        self.setLayout(mainLayout)

# if you call this file directly from python3, it will display a PyQt5 window
# good for confirming the parts of the assembly.

if __name__ == '__main__':
    main(model, tooltip, work, size=600, hud=None, lat=-75, lon=215)

```

10.4.9.10 Builtin Vismach Sample Panels

[QtVCP builtin Vismach Panels](#)

10.4.10 QtVCP Building Custom Widgets

10.4.10.1 Overview

Building custom widgets allows one to use the Qt Designer editor to place a custom widget rather than doing it manually in a handler file. A useful custom widgets would be a great way to contribute back to LinuxCNC.

Widget is the general name for the UI objects such as buttons and labels in pyQT. There are also special widgets made for LinuxCNC that make integration easier. This widgets can be placed with qt Designer editor - allowing one to see the result before actually loading the panel in LinuxCNC.

Qt Designer is a What You See is What You Get editor for placing pyQT widgets. It's original intend was for building the graphic widgets for programs. We leverage it to build screens and panels for LinuxCNC. In Qt Designer LinuxCNC widgets are split in three heading on the left side of the editor. One is for HAL only widgets. One is for LinuxCNC controller widgets And one is for dialog widgets.

For Designer to add custom widgets to it's editor it must have a plugin added to the right folder.

QtVCP does extra setup for widgets subclassed from `_HALWidgetBase`. (aka HAL-ified widgets)+ This includes injecting important variables, calling an extra setup function and calling a closing cleanup function at shutdown.

These functions are not called when the Designer editor displays the widgets. When QtVCP builds a screen from the .ui file, it searches for all the HAL-ified widgets. Then it finds the ScreenOptions widget, to collect information it needs to inject into the other widgets. It then instantiates each widget and if a HAL-fied widget, it calls the function `hal_init()`. `hal_init()` is defined in the base class and it adds variables such as the preference file to every HAL-ified widget. `hal_init()` will then call `_hal_init()` on the widget. `_hal_init()` allows the widget designer to do setup that requires access to the extra variables. Here is a description of the extra variables inject into HAL-ified widgets:

```
self.HAL_GCOMP = the HAL component instance
self.HAL_NAME = This widgets name as a string
self.QT_OBJECT_ = This widgets object instance
self.QTVCP_INSTANCE_ = The very toplevel Parent Of the screen
self.PATHS_ = The instance of QtVCP's path library
self.PREFS_ = the instance of an optional preference file
self.SETTINGS_ = the Qsettings object
```

When QtVCP closes, it calls the function `_hal_cleanup()` on all HAL-ified widgets. The baseclass creates an empty `_hal_cleanup()` function, which can be redefined in the custom widget subclass. This can be used to do such things as record preferences, etc. This function is not called when the Designer editor displays the widgets.

10.4.10.2 Custom Hal Widgets

Hal widgets are the simplest to show example of. `qtvcp/widgets/simple_widgets.py` holds many HAL only widgets. Lets look at a snippet of `simple_widgets.py`.

```
#!/usr/bin/env python3

#####
# Imports
#####
from PyQt5 import QtWidgets
from qtvcp.widgets.widget_baseclass import _HalWidgetBase, _HalSensitiveBase
import hal

#####
# WIDGET
#####

class Lcnc_GridLayout(QtWidgets.QWidget, _HalSensitiveBase):
    def __init__(self, parent = None):
        super(GridLayout, self).__init__(parent)
```

This is where we import libraries that our widget class needs.

In this case we need access to pyqt's QtWidgets library, LinuxCNC's HAL library and qtvcp's widget baseclass _HalSensitiveBase for automatic HAL pin setup and to disable/enable the widget (also known as input sensitivity).

There is also _HalToggleBase, and _HalScaleBase functions available in the library.

Ok, here is our custom widget based on pyQT's QGridLayout widget.

Grid layout allows one to place objects in a grid fashion.

But this grid_layout also will enable and disable all widgets inside it, based on the state of a HAL pin.

Line by Line:

```
class Lcnc_GridLayout(QtWidgets.QWidget, _HalSensitiveBase):
```

This defines the class name and the libraries it inherits from.

this class named Lcnc_GridLayout inherits the functions of QWidget and _HalSensitiveBase.

_HalSensitiveBase is *subclass* of _HalWidgetBase, the base class of most QtVCP widgets meaning it has all the functions of _HalWidgetBase plus the functions of _HalSensitiveBase.

It adds the function to make the widget be enabled or disabled based on a HAL input BIT pin.

```
def __init__(self, parent = None):
```

This is the function called when the widget is first made (said instantiated)- this is pretty standard.

```
super(GridLayout, self).__init__(parent)
```

This function initializes our widget's *Super* classes.

Super just means the inherited baseclasses; QWidget and _HalSensitiveBase

Pretty standard other than the widget name will change

10.4.10.3 Custom Controller Widgets using STATUS

Widget that interact with LinuxCNC's controller are only a little more complicated they require some extra libraries.

In this cut down example we will add properties that can be changed in Designer.

This LED indicator widget will respond to selectable LinuxCNC controller states.

```
#!/usr/bin/env python3
```

```
#####
# Imports
#####
from PyQt5.QtCore import pyqtProperty
from qtvcp.widgets.led_widget import LED
from qtvcp.core import Status

#####
# **** instantiate libraries section **** #
#####
STATUS = Status()

#####
# custom widget class definition
#####
class StateLED(LED):
    def __init__(self, parent=None):
        super(StateLED, self).__init__(parent)
        self.has_hal_pins = False
        self.setState(False)
        self.is_estopped = False
        self.is_on = False
        self.invert_state = False

    def _hal_init(self):
        if self.is_estopped:
            STATUS.connect('state-estop', lambda w:self._flip_state(True))
            STATUS.connect('state-estop-reset', lambda w:self._flip_state(False))
        elif self.is_on:
            STATUS.connect('state-on', lambda w:self._flip_state(True))
            STATUS.connect('state-off', lambda w:self._flip_state(False))

    def _flip_state(self, data):
        if self.invert_state:
            data = not data
        self.change_state(data)

#####
# Designer properties setter/getters/resetters
#####

# invert status
def set_invert_state(self, data):
    self.invert_state = data
def get_invert_state(self):
    return self.invert_state
def reset_invert_state(self):
    self.invert_state = False

# machine is estopped status
def set_is_estopped(self, data):
    self.is_estopped = data
def get_is_estopped(self):
    return self.is_estopped
def reset_is_estopped(self):
    self.is_estopped = False

# machine is on status
def set_is_on(self, data):
    self.is_on = data
def get_is_on(self):
    return self.is_on
```

```
def reset_is_on(self):
    self.is_on = False

#####
# Designer properties
#####
invert_state_status = pyqtProperty(bool, get_invert_state, set_invert_state, ←
    reset_invert_state)
is_estopped_status = pyqtProperty(bool, get_is_estopped, set_is_estopped, ←
    reset_is_estopped)
is_on_status = pyqtProperty(bool, get_is_on, set_is_on, reset_is_on)
```

This is where we import libraries that our widget class needs.
 We import pyqtProperty so we can interact with the Designer editor.
 we import LED because our custom widget is based on it.
 We import Status because it gives us status messages from LinuxCNC.

Typically we instantiated the libraries outside of the widget class so that the reference to it is global - meaning you don't need to use self. in front of it.
 By convention we use all capital letters in the name.

This is the meat and potatoes of our custom widget.

```
class StateLed(LED):
    def __init__(self, parent=None):
        super(StateLed, self).__init__(parent)
        self.has_hal_pins = False
        self.setState(False)
        self.is_estopped = False
        self.is_on = False
        self.invert_state = False
```

This defines the name of our custom widget and what other class it inherits from, in this case we inherit LED - a QtVCP widget that represents a status light.

The *init* is typical of most widgets, it is called when the widget is first made.

The super line is typical of most widgets - it calls the parent (super) widget's initialization code.

Then we set some attributes.

self.has_hal_pins is an attribute inherited from Lcnc_Led - we set it here so no HAL Pins are made.

self.setState is inherited from Lcnc_led - we set it to make sure the LED is off.

The other attributes are for the selectable options of our widget.

```
def _hal_init(self):
    if self.is_estopped:
        STATUS.connect('state-estop', lambda w:self._flip_state(True))
        STATUS.connect('state-estop-reset', lambda w:self._flip_state(False))
    elif self.is_on:
        STATUS.connect('state-on', lambda w:self._flip_state(True))
        STATUS.connect('state-off', lambda w:self._flip_state(False))
```

This function connects STATUS (LinuxCNC status message library) to our widget so that the LED will on or off based on the selected state of the controller. We have two states we can choose from is_estopped or is_on.

Depending on which is active our widget get connected to the appropriate STATUS messages.

_hal_int() is called on each widget that inherited _HalWidgetBase, when QtVCP first builds the screen.

You might wonder why it's called on this widget since we didn't have _HalWidgetBase in our class definition (class Lcnc_State_Led(Lcnc_Led):) - it's called because Lcnc_Led inherits _HalWidgetBase

in this function you have access to some extra information. (though we don't use them in this example)

```

self.HAL_GCOMP = the HAL component instance
self.HAL_NAME = This widgets name as a string
self.QT_OBJECT_ = This widgets PyQt object instance
self.QTVCP_INSTANCE_ = The very toplevel Parent Of the screen
self.PATHS_ = The instance of QtVCP's path library
self.PREFS_ = the instance of an optional preference file

```

We could use this information to create HAL pins or look up image paths etc.

```
STATUS.connect('state-estop', lambda w:self._flip_state(True))
```

lets look at this line more closely. STATUS is very common theme is widget building. STATUS use GObject message system to send messages to widgets that register to it. This line is the register process.

state-estop is the message we wish to act on. there are many messages available.

lambda w:self._flip_state(True) is what happens when the message is caught.

the lambda function accepts the widget instance (w) that GObject sends it and then calls the function *self._flip_state(True)*

Lambda was used to strip the (w) object before calling the *self._flip_state* function.

It also allowed use to send *self._flip_state()* the True state.

```

def _flip_state(self, data):
    if self.invert_state:
        data = not data
    self.change_state(data)

```

This is the function that actually flips the state of the LED.

It is what gets called when the appropriate STATUS message is accepted.

```
STATUS.connect('current-feed-rate', self._set_feedrate_text)
```

and the function called looks like this:

```
def _set_feedrate_text(self, widget, data):
```

in which the widget and any data must be accepted by the function.

This is how Designer sets the attributes of the widget.

This can also be called directly in the widget.

This is the registering of properties in Qt Designer.

The property name is the text that is used in Qt Designer.

These property names cannot be the same as the attributes they represent.

These properties show in Designer in the order they appear here.

10.4.10.4 Custom Controller Widgets with actions

Here is an example of a widget that sets the user reference system.

It changes the machine controller state with the ACTION library.

It also uses the STATUS library to set whether the button can be clicked or not.

```

import os
import hal

from PyQt5.QtWidgets import QWidget, QPushButton, QMenu, QAction
from PyQt5.QtCore import Qt, QEvent, pyqtProperty, QBasicTimer, pyqtSignal
from PyQt5.QtGui import QIcon

```

```

from qtvcp.widgets.widget_baseclass import _HalWidgetBase
from qtvcp.widgets.dialog_widget import EntryDialog
from qtvcp.core import Status, Action, Info

# Instantiate the libraries with global reference
# STATUS gives us status messages from LinuxCNC
# INFO holds ini details
# ACTION gives commands to LinuxCNC
STATUS = Status()
INFO = Info()
ACTION = Action()

class SystemToolButton(QToolButton, _HalWidgetBase):
    def __init__(self, parent=None):
        super(SystemToolButton, self).__init__(parent)
        self._joint = 0
        self._last = 0
        self._block_signal = False
        self._auto_label_flag = True
        SettingMenu = QMenu()
        for system in ('G54', 'G55', 'G56', 'G57', 'G58', 'G59', 'G59.1', 'G59.2', 'G59.3'):

            Button = QAction(QIcon('exit24.png'), system, self)
            Button.triggered.connect(self[system.replace('.', '_')])
            SettingMenu.addAction(Button)

        self.setMenu(SettingMenu)
        self.dialog = EntryDialog()

    def _hal_init(self):
        if not self.text() == '':
            self._auto_label_flag = False
        def homed_on_test():
            return (STATUS.machine_is_on()
                    and (STATUS.is_all_homed() or INFO.NO_HOME_REQUIRED))

        STATUS.connect('state-off', lambda w: self.setEnabled(False))
        STATUS.connect('state-estop', lambda w: self.setEnabled(False))
        STATUS.connect('interp-idle', lambda w: self.setEnabled(homed_on_test()))
        STATUS.connect('interp-run', lambda w: self.setEnabled(False))
        STATUS.connect('all-homed', lambda w: self.setEnabled(True))
        STATUS.connect('not-all-homed', lambda w, data: self.setEnabled(False))
        STATUS.connect('interp-paused', lambda w: self.setEnabled(True))
        STATUS.connect('user-system-changed', self._set_user_system_text)

    def G54(self):
        ACTION.SET_USER_SYSTEM('54')

    def G55(self):
        ACTION.SET_USER_SYSTEM('55')

    def G56(self):
        ACTION.SET_USER_SYSTEM('56')

    def G57(self):
        ACTION.SET_USER_SYSTEM('57')

    def G58(self):
        ACTION.SET_USER_SYSTEM('58')

    def G59(self):

```

```

ACTION.SET_USER_SYSTEM('59')

def G59_1(self):
    ACTION.SET_USER_SYSTEM('59.1')

def G59_2(self):
    ACTION.SET_USER_SYSTEM('59.2')

def G59_3(self):
    ACTION.SET_USER_SYSTEM('59.3')

def _set_user_system_text(self, w, data):
    convert = { 1:"G54", 2:"G55", 3:"G56", 4:"G57", 5:"G58", 6:"G59", 7:"G59.1", 8:"G59 ←
               .2", 9:"G59.3"}
    if self._auto_label_flag:
        self.setText(convert[int(data)])

def ChangeState(self, joint):
    if int(joint) != self._joint:
        self._block_signal = True
        self.setChecked(False)
        self._block_signal = False
        self.hal_pin.set(False)

#####
# required class boiler code #
#####

def __getitem__(self, item):
    return getattr(self, item)
def __setitem__(self, item, value):
    return setattr(self, item, value)

```

10.4.10.5 Stylesheet Property Changes Based On Events

It's possible to have widgets restyled when events change.
 You must explicitly *polish* the widget to have PyQt redo the style.
 This is a relatively expensive function so should be used sparingly.
 This example will set the property `isHomed` based on LinuxCNC's homed state.
 This property can be used in the stylesheet to change stylesheet properties.

```

class HomeLabel(QLabel, _HalWidgetBase):
    def __init__(self, parent=None):
        super(HomeLabel, self).__init__(parent)
        self.joint_number = 0
        # for stylesheet reading
        self._isHomed = False

    def _hal_init(self):
        super(HomeLabel, self)._hal_init()
        STATUS.connect('homed', lambda w,d: self._home_status_polish(int(d), True))
        STATUS.connect('unhomed', lambda w,d: self._home_status_polish(int(d), False))

    # update ishomed property
    # polish widget so stylesheet sees the property change
    # some stylesheets color the text on home/unhome
    def _home_status_polish(self, d, state):
        if self.joint_number == d:
            self.setProperty('isHomed', state)
            self.style().unpolish(self)

```

```

        self.style().polish(self)

# Qproperty getter and setter
def getisHomed(self):
    return self._isHomed
def setisHomed(self, data):
    self._isHomed = data

# Qproperty
isHomed = QtCore.pyqtProperty(bool, getisHomed, setisHomed)

```

Here is a sample stylesheet to change text color based on home state.

In this case any widget based on the HomeLabel widget above will change text color.

You would usually pick specific widgets using *HomeLabel #specific_widget_name[homed=true]{*

```

HomeLabel[homed=true] {
    color: green;
}
HomeLabel[homed=false] {
    color: red;
}

```

10.4.10.6 Use Stylesheets To Change Custom Widget Properties

```

class Label(QLabel):
    def __init__(self, parent=None):
        super(Label, self).__init__(parent)
        alternateFont0 = self.font

# Qproperty getter and setter
def getFont0(self):
    return self.alternateFont0
def setFont0(self, value):
    self.alternateFont0(value)
# Qproperty
styleFont0 = pyqtProperty(QFont, getFont0, setFont0)

```

Sample stylesheet that sets a custom widget property.

```

Label{
qproperty-styleFont0: "Times,12,-1,0,90,0,0,0,0,0";
}

```

10.4.10.7 Widget Plugins

We must register our custom widget for Designer to use them.

Here is a typical samples

they would need to be added to qtvcp/plugins/

Then qtvcp/plugins/qtvcplugin.py would need to be adjusted to import them.

```

#!/usr/bin/env python3

from PyQt5 import QtCore, QtGui
from PyQt5.QtDesigner import QPyDesignerCustomWidgetPlugin
from qtvcp.widgets.simple_widgets import Lcnc_GridLayout
from qtvcp.widgets.qtvcp_icons import Icon
ICON = Icon()

```



```
#####
# GridLayout
#####
class LcncGridLayoutPlugin(QPyDesignerCustomWidgetPlugin):
    def __init__(self, parent = None):
        QPyDesignerCustomWidgetPlugin.__init__(self)
        self.initialized = False
    def initialize(self, formEditor):
        if self.initialized:
            return
        self.initialized = True
    def isInitialized(self):
        return self.initialized
    def createWidget(self, parent):
        return Lcnc_GridLayout(parent)
    def name(self):
        return "Lcnc_GridLayout"
    def group(self):
        return "LinuxCNC - HAL"
    def icon(self):
        return QtGui.QIcon(QtGui.QPixmap(ICON.get_path('lcnc_gridlayout')))
    def toolTip(self):
        return "HAL enable/disable GridLayout widget"
    def whatsThis(self):
        return ""
    def isContainer(self):
        return True
    def domXml(self):
        return '<widget class="Lcnc_GridLayout" name="lcnc_gridlayout" />\n'
    def includeFile(self):
        return "qtvcp.widgets.simple_widgets"
```

```
#!/usr/bin/env python3
```

```
from PyQt5 import QtCore, QtGui
from PyQt5.QtDesigner import QPyDesignerCustomWidgetPlugin
from qtvcp.widgets.system_tool_button import SystemToolButton
from qtvcp.widgets.qtvcp_icons import Icon
ICON = Icon()

#####
# SystemToolButton
#####
class SystemToolButtonPlugin(QPyDesignerCustomWidgetPlugin):
    def __init__(self, parent = None):
        super(SystemToolButtonPlugin, self).__init__(parent)
        self.initialized = False
    def initialize(self, formEditor):
        if self.initialized:
            return
        self.initialized = True
    def isInitialized(self):
        return self.initialized
    def createWidget(self, parent):
        return SystemToolButton(parent)
    def name(self):
        return "SystemToolButton"
    def group(self):
        return "LinuxCNC - Controller"
    def icon(self):
        return QtGui.QIcon(QtGui.QPixmap(ICON.get_path('systemtoolbutton')))
```

```

def toolTip(self):
    return "Button for selecting a User Coordinate System"
def whatsThis(self):
    return ""
def isContainer(self):
    return False
def domXml(self):
    return '<widget class="SystemToolButton" name="systemtoolbutton" />\n'
def includeFile(self):
    return "qtvcp.widgets.system_tool_button"

```

It possible to add an entry to the dialog that pops up when you right click the widget in the layout. This can do such things as select options in a more convenient way. This is the plugin used for action buttons.

```

#!/usr/bin/env python3

import sip
from PyQt5 import QtCore, QtGui, QtWidgets
from PyQt5.QtDesigner import QPyDesignerCustomWidgetPlugin, \
    QPyDesignerTaskMenuExtension, QExtensionFactory, \
    QDesignerFormWindowInterface, QPyDesignerMemberSheetExtension
from qtvcp.widgets.action_button import ActionButton
from qtvcp.widgets.qtvcp_icons import Icon
ICON = Icon()

Q_TYPEID = {
    'QDesignerContainerExtension': 'org.qt-project.Qt.Designer.Container',
    'QDesignerPropertySheetExtension': 'org.qt-project.Qt.Designer.PropertySheet',
    'QDesignerTaskMenuExtension': 'org.qt-project.Qt.Designer.TaskMenu',
    'QDesignerMemberSheetExtension': 'org.qt-project.Qt.Designer.MemberSheet'
}

#####
# ActionBUTTON
#####
class ActionButtonPlugin(QPyDesignerCustomWidgetPlugin):

    # The __init__() method is only used to set up the plugin and define its
    # initialized variable.
    def __init__(self, parent=None):
        super(ActionButtonPlugin, self).__init__(parent)
        self.initialized = False

    # The initialize() and isInitialized() methods allow the plugin to set up
    # any required resources, ensuring that this can only happen once for each
    # plugin.
    def initialize(self, formEditor):

        if self.initialized:
            return
        manager = formEditor.extensionManager()
        if manager:
            self.factory = ActionButtonTaskMenuFactory(manager)
            manager.registerExtensions(self.factory, Q_TYPEID['QDesignerTaskMenuExtension'] ←
                ])
            self.initialized = True

    def isInitialized(self):
        return self.initialized

```

```
# This factory method creates new instances of our custom widget
def createWidget(self, parent):
    return QPushButton(parent)

# This method returns the name of the custom widget class
def name(self):
    return "ActionButton"

# Returns the name of the group in Qt Designer's widget box
def group(self):
    return "LinuxCNC - Controller"

# Returns the icon
def icon(self):
    return QtGui.QIcon(QtGui.QPixmap(ICON.get_path('actionbutton'))))

# Returns a tool tip short description
def toolTip(self):
    return "Action button widget"

# Returns a short description of the custom widget for use in a "What's
# This?" help message for the widget.
def whatsThis(self):
    return ""

# Returns True if the custom widget acts as a container for other widgets;
def isContainer(self):
    return False

# Returns an XML description of a custom widget instance that describes
# default values for its properties.
def domXml(self):
    return '<widget class="ActionButton" name="actionbutton" />\n'

# Returns the module containing the custom widget class. It may include
# a module path.
def includeFile(self):
    return "qtvcp.widgets.action_button"

class ActionButtonDialog(QtWidgets.QDialog):

    def __init__(self, widget, parent = None):

        QtWidgets.QDialog.__init__(self, parent)

        self.widget = widget

        self.previewWidget = ActionButton()

        buttonBox = QtWidgets.QDialogButtonBox()
        okButton = buttonBox.addButton(buttonBox.Ok)
        cancelButton = buttonBox.addButton(buttonBox.Cancel)

        okButton.clicked.connect(self.updateWidget)
        cancelButton.clicked.connect(self.reject)

        layout = QtWidgets.QGridLayout()
        self.c_estop = QtWidgets.QCheckBox("Estop Action")
        self.c_estop.setChecked(widget.estop )
        layout.addWidget(self.c_estop)
```

```

        layout.addWidget(buttonBox, 5, 0, 1, 2)
        self.setLayout(layout)

        self.setWindowTitle(self.tr("Set Options"))

    def updateWidget(self):

        formWindow = QDesignerFormWindowInterface.findFormWindow(self.widget)
        if formWindow:
            formWindow.cursor().setProperty("estop_action",
                QtCore.QVariant(self.c_estop.isChecked()))
        self.accept()

class ActionButtonMenuEntry(QPyDesignerTaskMenuExtension):

    def __init__(self, widget, parent):
        super(QPyDesignerTaskMenuExtension, self).__init__(parent)
        self.widget = widget
        self.editStateAction = QtWidgets.QAction(
            self.tr("Set Options..."), self)
        self.editStateAction.triggered.connect(self.updateOptions)

    def preferredEditAction(self):
        return self.editStateAction

    def taskActions(self):
        return [self.editStateAction]

    def updateOptions(self):
        dialog = ActionButtonDialog(self.widget)
        dialog.exec_()

class ActionButtonTaskMenuFactory(QExtensionFactory):
    def __init__(self, parent = None):
        QExtensionFactory.__init__(self, parent)

    def createExtension(self, obj, iid, parent):

        if not isinstance(obj, ActionButton):
            return None
        if iid == Q_TYPEID['QDesignerTaskMenuExtension']:
            return ActionButtonMenuEntry(obj, parent)
        elif iid == Q_TYPEID['QDesignerMemberSheetExtension']:
            return ActionButtonMemberSheet(obj, parent)
        return None

```

10.4.11 QtVCP Handler file code snippets

Here are bits of ideas to put in the handler file.

10.4.11.1 Preference file loading/saving

Here is how to load and save at closing time a number and some text: You must have included a preference file option in the screenoptions widget. The preference file path is set in the INI.

Under the *def initialized__(self):* function add:

```

if self.w.PREFS_:
    # variable name (entry name, default value, type, section name)

```

```
self.int_value = self.w.PREFS_.getpref('Integer_value', 75, int, 'CUSTOM_FORM_ENTRIES')
self.string_value = self.w.PREFS_.getpref('String_value', 'on', str, ' ←
CUSTOM_FORM_ENTRIES')
```

Under the *def closing_cleanup__(self):* function add:

```
if self.w.PREFS_:
    # variable name (entry name, variable name, type, section name)
    self.w.PREFS_.putpref('Integer_value', self.integer_value, int, 'CUSTOM_FORM_ENTRIES')
    self.w.PREFS_.putpref('String_value', self.string_value, str, 'CUSTOM_FORM_ENTRIES')
```

10.4.11.2 use QSettings to read/save variable

Here is how to load and save variables using PyQt's QSettings functions:

- The file is actually saved in `~/.config/QtVcp`
- You can set a Group name to help keep names organized and unique.
- When reading a setting, if there is no entry, it will return None.
- You should set a default to cover the first time it is run.

In this example:

- We added *or 20* and *or 2.5* as defaults.
- The names *MyGroupName*, *int_value*, *float_value*, *myInteger*, and *myFloat* are user defined.

Under the *def initialized__(self):* function add:

```
# set recorded columns sort settings
self.SETTINGS_.beginGroup("MyGroupName")
self.int_value = self.SETTINGS_.value('myInteger', type = int) or 20
self.float_value = self.SETTINGS_.value('myFloat', type = float) or 2.5
self.SETTINGS_.endGroup()
```

Under the *def closing_cleanup__(self):* function add:

```
# save values with QSettings
self.SETTINGS_.beginGroup("MyGroupName")
self.SETTINGS_.setValue('myInteger', self.int_value)
self.SETTINGS_.setValue('myFloat', self.float_value)
self.SETTINGS_.endGroup()
```

10.4.11.3 Add a basic style editor

Being able to edit a style on a running screen is convenient.

In the *IMPORT SECTION*:

```
from qtvcp.widgets.stylesheeteditor import StyleSheetEditor as SSE

STYLEEDITOR = SSE()
```

In the *INITIALIZE SECTION*, under the `__init__(self, halcomp, widgets, paths):` function:

```
KEYBIND.add_call('Key_F12', 'on_keycall_F12')
```

Finally lets make F12 launch it.
In the **KEYBINDING SECTION** add:

```
def on_keycall_F12(self,event,state,shift,cntrl):
    if state:
        STYLEEDITOR.load_dialog()
```

10.4.11.4 Request Dialog Entry

QtVCP uses STATUS messages to pop up and return information from dialogs. Prebuilt dialogs keep track of their last position and include options for focus shading and sound. To get information back from the dialog requires using a STATUS general message.

In the *IMPORT SECTION* make sure there is an entry similar to this:

```
from qtvcp.core import Status
STATUS = Status()
```

This loads and initializes the STATUS library.

In the *INITIALIZE SECTION* Under the `__init__.(self, halcomp, widgets, paths):` function

```
STATUS.connect('general',self.return_value)
```

This registers STATUS to call the function `self.return_value` when a general message is sent.

In the *GENERAL FUNCTIONS SECTION*

```
def request_number(self):
    mess = {'NAME':'ENTRY','ID':'FORM__NUMBER', 'TITLE':'Set Tool Offset'}
    STATUS.emit('dialog-request', mess)
```

This is the function to request an entry dialog.

NAME needs to be set to the dialogs unique launch name.

ID needs to be set to a unique name that the function supplies

It creates a python dict. The NAME sets which dialog to request - *ENTRY* or *CALCULATOR* allows entering numbers.

The ID should be a unique key. TITLE sets the dialog title. You can also add arbitrary data to the dict -+ the dialog will ignore them but send them back to the return code.

In the *CALLBACKS FROM STATUS SECTION*

```
# process the STATUS return message from set-tool-offset
def return_value(self, w, message):
    num = message.get('RETURN')
    id_code = bool(message.get('ID') == 'FORM__NUMBER')
    name = bool(message.get('NAME') == 'ENTRY')
    if id_code and name and num is not None:
        print('The {} number from {} was: {}'.format(name, id_code, num))
```

This catches all general messages so must check the dialog type and id code to confirm it's our dialog. In this case we had requested an *ENTRY* dialog and our unique id was *ENTRY_NUMBER*, so now we know the message is for us.

Entry or Calculator dialogs return a float number.

10.4.11.5 Speak a Startup Greeting

This requires the *espeak* library installed on the system.

In the *IMPORT SECTION* make sure there is an entry similar to this:

```
from qtvcp.core import Status
STATUS = Status()
```

In the *INITIALIZE SECTION* Under the `__init__(self, halcomp, widgets, paths):` function

```
STATUS.emit('play-alert', 'SPEAK Please remember to oil the ways.')
```

SPEAK is a key work, everything after it will be pronounced

10.4.11.6 ToolBar Functions.

ToolBar buttons and submenus are added in Designer but the code to make them do something is added in the handler file.

In this example we assume you added a tool bar with one submenu and three actions.

These will be configure to create a recent file selection menu, an about pop up dialog action, a quit program action and a user defined function action.

You can add submenus in designer by adding an qaction (by typing in the toolbar column) then clicking the *plus* icon on the right.

This will ad a sub column that you need to type a name into. Now the original Qaction will be a Qmenu instead.

Now erase the Qaction you added to that Qmenu - the menu will stay as a menu.

The objectName of the toolbar button is used to identify the button when configuring it - descriptive names help.

Using the action editor menu, right click and select edit. Edit the object name, text, and button type for an appropriate action.

In this example the submenu name must be : *menuRecent*. The actions must be *actionAbout*, *action-Quit*, *actionMyFunction*

In the *IMPORT SECTION* add:

```
from qtvcp.lib.toolbar_actions import ToolBarActions
```

Loads the toolbar library.

in the *INSTANTIATE LIBRARY* Section add:

```
TOOLBAR = ToolBarActions()
```

In the *SPECIAL FUNCTIONS SECTION* Under the `def initialized__(self):` function add:

```
TOOLBAR.configure_submenu(self.w.menuRecent, 'recent_submenu')
TOOLBAR.configure_action(self.w.actionAbout, 'about')
TOOLBAR.configure_action(self.w.actionQuit, 'Quit', lambda d:self.w.close())
TOOLBAR.configure_action(self.w.actionMyFunction, 'My Function', self.my_function)
```

Configures the action.

In the *GENERAL FUNCTIONS SECTION* ADD:

```
def my_function(self, widget, state):
    print('My function State = {}'.format(state))
```

The function to be called if the actionMyFunction button is pressed.

10.4.11.7 Add HAL Pins that call functions

In this way you don't need to poll the state of input pins.

In the *IMPORT SECTION* add:

```
from qtvcp.core import Qhal
```

Loads the Qhal library for access to QtVCP's HAL component.

In the *INstantiate LIBRARY* Section add:

```
QHAL = Qhal()
```

Under the initialised__ function, make sure there is an entry similar to this:

```
#####
# Special Functions called from QtVCP
#####

# at this point:
# the widgets are instantiated.
# the HAL pins are built but HAL is not set ready
def initialized__(self):
    self.pin_cycle_start_in = QHAL.newpin('cycle-start-in', QHAL.HAL_BIT, QHAL.HAL_IN)
    self.pin_cycle_start_in.value_changed.connect(lambda s: self.cycleStart(s))
```

Add a function that gets called when the pin state changes. This function assumes there is a Tab widget named *mainTab* that has tabs with the names *tab_auto*, *tab_graphics*, *tab_filemanager* and *tab_mdi*.

In this way, the cycle start button works differently depending on what tab is showing. This is simplified - checking state and error trapping might be helpful.

In the *GENERAL FUNCTIONS SECTION* add:

```
#####
# general functions #
#####

def cycleStart(self, state):
    if state:
        tab = self.w.mainTab.currentWidget()
        if tab in( self.w.tab_auto, self.w.tab_graphics):
            ACTION.RUN(line=0)
        elif tab == self.w.tab_files:
            self.w.filemanager.load()
        elif tab == self.w.tab_mdi:
            self.w.mditouchy.run_command()
```

10.4.11.8 Add a special Max Velocity Slider based on percent

Some times you want to build a widget to do something not built in.

The built in Max velocity slider acts on units per minute, here we show how to do percent:

The STATUS command makes sure the slider adjusts if linuxcnc changes the current max velocity. `valueChanged.connect()` calls a function when the slider is moved.

In Designer add a QSlider widget called *mvPercent*. Then add the code to the handler file.


```
#####
# SPECIAL FUNCTIONS SECTION #
#####

def initialized__(self):
    self.w.mvPercent.setMaximum(100)
    STATUS.connect('max-velocity-override-changed', lambda w, data: self.w.mvPercent. ←
        setValue((data / INFO.MAX_TRAJ_VELOCITY)*100))
    self.w.mvPercent.valueChanged.connect(self.setMVPercentValue)

#####
# GENERAL FUNCTIONS #
#####

def setMVPercentValue(self, value):
    ACTION.SET_MAX_VELOCITY_RATE(INFO.MAX_TRAJ_VELOCITY * (value/100.0))
```

10.4.11.9 Toggle Continuous Jog On and Off

Generally selecting continuous jogging is a momentary button, that requires you to select the previous jog increment after. We will build a button that toggles between continuous jog and whatever increment that was already selected.

In Designer: - Use an action button with no action, call it *btn_toggle_continuous*. - Set the AbstractButton property *checkable* true. - Set the ActionButton properties *incr_imperial_number* and *incr_mm_number* to 0.

Use designer's slot editor to use the button signal *clicked(bool)* to call form's handler function *toggle_continuous_clicked()*, see [Using Designer to add slots](#).

Then add this code snippets to the handler file:

Under the *initialized__* function, make sure there is an entry similar to this:

```
# at this point:
# the widgets are instantiated.
# the HAL pins are built but HAL is not set ready
def initialized__(self):
    STATUS.connect('jogincrement-changed', lambda w, d, t: self.record_jog_incr(d,t))
    # set a default increment to toggle back to
    self.L_incr = 0.01
    self.L_text = "0.01in"
```

In the *GENERAL FUNCTIONS SECTION* ADD:

```
#####
# general functions #
#####

# if it isn't continuous, record the latest jog increment
# and untoggle the continuous button
def record_jog_incr(self,d, t):
    if d != 0:
        self.L_incr = d
        self.L_text = t
        self.w.btn_toggle_continuous.safecheck(False)
```

In the *CALLBACKS FROM FORM SECTION* ADD:

```
#####
# CALLBACKS FROM FORM #
```

```
#####
def toggle_continuous_clicked(self, state):
    if state:
        # set continuous (call the actionbutton's function)
        self.w.btn_toggle_continuous.incr_action()
    else:
        # reset previously recorded increment
        ACTION.SET_JOG_INCR(self.L_incr, self.L_text)
```

10.4.11.10 Class Patch the file manager widget

Note

Class patching (monkey patching) is a little like black magic - so use it only if needed.

The File manager widget is designed to load a selected program in LinuxCNC. But maybe you want to print the file name first. We can *class patch* the library to redirect the function call.

In the *IMPORT SECTION* add:

```
from qtvcp.widgets.file_manager import FileManager as FM
```

Here we are going to keep a reference to the original function, so we can still call it. Then we redirect the class to call our custom function in the handler file instead.

```
#####
# Special Functions called from QtVCP
#####

# For changing functions in widgets we can 'class patch'.
# class patching must be done before the class is instantiated.
def class_patch__(self):
    self.old_load = FM.load # keep a reference of the old function
    FM.load = self.our_load # redirect function to our handle file function
```

Ok Now we write a custom function to replace the original. This function must have the same signature as the original function. In this example we are still going to call the original function by using the reference to it we recorded earlier. It requires the first argument to be the widget instance which in this case is `self.w.filemanager` (the name given in the designer editor).

```
#####
# GENERAL FUNCTIONS #
#####

def our_load(self, fname):
    print(fname)
    self.old_load(self.w.filemanager, fname)
```

Now our custom function will print the file path to the terminal before loading the file. Obviously boring but shows the principle.

There is another slightly different way to do this that can have advantages. You can store the reference to the original function in the original class. the trick here is to make sure the function name you use to store it, is not already used in the class. *super__* added to the function name would be a good choice. We won't use that in built in QtVCP widgets.

```
#####
# Special Functions called from QtVCP
#####

# For changing functions in widgets we can 'class patch'.
# class patching must be done before the class is instantiated.
def class_patch__(self):
    FM.super__load = FM.load # keep a reference of the old function in the original class
    FM.load = self.our_load # redirect function to our handle file function

#####
# GENERAL FUNCTIONS #
#####

def our_load(self, fname):
    print(fname)
    self.w.filemanager.super__load(fname)
```

10.4.11.11 Adding widgets Programmatically

In some situation it is only possible to add widgets with python code rather than using the Designer editor. When adding QtVCP widgets programmatically, sometimes there are extra steps to be taken. Here we are going to add a spindle speed indicator bar and up-to-speed LED to a tab widget corner. Qt Designer does not support adding corner widgets to tabs but PyQt does. This is a cut down example from Qtaxis screen's handler file.

First we must import the libraries we need. Often these libraries are already imported in the handler file: - QtWidgets gives us access to the QProgress bar. - QColor is for the LED color. - StateLED is the QtVCP library used to create the spindle-at-speed LED. - Status is used to catch LinuxCNC status information. - Info gives us information about the machine configuration.

```
#####
# **** IMPORT SECTION **** #
#####

from PyQt5 import QtWidgets
from PyQt5.QtGui import QColor
from qtvcp.widgets.state_led import StateLED as LED
from qtvcp.core import Status, Info
```

STATUS and INFO are initialized outside the handler class so as to be a global reference (no self. in front).

```
#####
# **** instantiate libraries section **** #
#####

STATUS = Status()
INFO = Info()
```

For the spindle speed indicator we need to know the current spindle speed: We register with STATUS to catch the *actual-spindle-speed-changed* signal to call a function named: *self.update_spindle()*.

```
#####
# **** INITIALIZE **** #
#####
# widgets allows access to widgets from the QtVCP files
# at this point the widgets and hal pins are not instantiated
def __init__(self, halcomp, widgets, paths):
```

```

self.hal = halcomp
self.w = widgets
self.PATHS = paths

STATUS.connect('actual-spindle-speed-changed', lambda w,speed: self.update_spindle(↔
    speed))

```

We need to make sure the Designer widgets are already built before we try to add to them. We add a function call `self.make_corner_widgets()` to build our extra widgets at the right time.

```

#####
# Special Functions called from QTSCREEN
#####

# at this point:
# the widgets are instantiated.
# the HAL pins are built but HAL is not set ready
def initialized__(self):
    self.make_corner_widgets()

```

Ok let's code the function to build the widgets and add them in the tab widget.

We are assuming there is a tab widget built with Designer called *rightTab*.

`self.w.led = LED()` - this initializes the basic StateLed widget and uses `self.w.led` as the reference from then on.

`self.w.led.setProperty("is_spindle_at_speed_status", True)` - since the state LED can be used for many indications

we must set the property that designates it as a spindle-at-speed LED.

`self.w.led.setProperty("color", QColor(0,255,0,255))` this sets it as green when on.

`self.w.led.hal_init(HAL_NAME = "spindle_is_at_speed")` - this is the extra function call required with some QtVCP widgets.

If `HAL_NAME` is omitted it will use the widget objectName if there is one.

It gives the special widgets reference to:

- `self.HAL_GCOMP_` - The HAL component wrapped in QtVCP's core QComponent
- `self.HAL_NAME_` -The HAL widget name
- `self.QT_OBJECT_` -the actual object
- `self.QTVCP_INSTANCE_` - The window object
- `self.PATHS_` -the path library
- `self.PREFS_` -the preference object.

`self.w.rpm_bar = QtWidgets.QProgressBar()` - initialize a PyQt5 QProgress bar.

`self.w.rpm_bar.setRange(0, INFO.MAX_SPINDLE_SPEED)` - set the max range of the Progress bar to the max specified in the INI.

Since you can only add one widget to the tab corner and we have two we want there, we must add the two into a container.

We create a QWidget and add a QHBoxLayout to the QWidget.

The we add our QProgress bar and LED to the layout.

`self.w.rightTab.setCornerWidget(w)` - finally we add the QWidget (with our QProgress bar and LED in it) to the tab widget's corner.

```

#####
# general functions #
#####

```

```
def make_corner_widgets(self):
    # make a spindle-at-speed green LED
    self.w.led = LED()
    self.w.led.setProperty('is_spindle_at_speed_status', True)
    self.w.led.setProperty('color', QColor(0, 255, 0, 255))
    self.w.led.hal_init(HAL_NAME = 'spindle_is_at_speed')

    # make a spindle speed bar
    self.w.rpm_bar = QtWidgets.QProgressBar()
    self.w.rpm_bar.setRange(0, INFO.MAX_SPINDLE_SPEED)

    # container
    w = QtWidgets.QWidget()
    w.setContentsMargins(0, 0, 0, 6)
    w.setMinimumHeight(40)

    # layout
    hbox = QtWidgets.QHBoxLayout()
    hbox.addWidget(self.w.rpm_bar)
    hbox.addWidget(self.w.led)
    w.setLayout(hbox)

    # add the container to the corner of the right tab widget
    self.w.rightTab.setCornerWidget(w)
```

Now we build the function to actually update our QProgressBar when STATUS updates the spindle speed.

self.w.rpm_bar.setInvertedAppearance() - In this case we chose to display left-to-right or right-to-left depending if we are turning clockwise or anticlockwise.

self.w.rpm_bar.setFormat() - This formats the writing in the bar.

self.w.rpm_bar.setValue() - This sets the length of the colored bar.

```
#####
# callbacks from STATUS #
#####
def update_spindle(self, data):
    self.w.rpm_bar.setInvertedAppearance(bool(data<0))
    self.w.rpm_bar.setFormat('{0:d} RPM'.format(int(data)))
    self.w.rpm_bar.setValue(abs(data))
```

10.4.11.12 Update/Read Objects Periodically

Sometimes you need to update a widget or read a value regularly that isn't covered by normal libraries.

Here we update an LED based on a watched HAL pin every 100ms.

We assume there is an LED named *led* in the designer .ui file.

In the *IMPORT SECTION* add:

```
from qtvcp.core import Qhal
```

Loads the Qhal library for access to QtVCP's hal component.

in the *INstantiate LIBRARY* Section add:

```
QHAL = Qhal()
```

Now add/modify these sections to include code that is similar to this:

```
#####
# **** INITIALIZE **** #
#####
# widgets allows access to widgets from the QtVCP files
# at this point the widgets and hal pins are not instantiated
def __init__(self, halcomp, widgets, paths):
    self.hal = halcomp
    self.w = widgets
    self.PATHS = paths

    # register a function to be called at CYCLE_TIME period (usually every 100ms)
    STATUS.connect('periodic', lambda w: self.update_periodic())

#####
# general functions #
#####
def update_periodic(self):
    data = QHAL.getvalue('spindle.0.is-oriented')
    self.w.led.setState(data)
```

10.4.11.13 external control with ZMQ messaging reading

Sometimes you want to control the screen with a separate program. QtVCP can automatically set up ZMQ messaging to send and/or receive remote messages. It uses ZMQ's publish/subscribe pattern of messages. As always consider security before letting programs interface through messaging. In the screenoptions widget, you can select the property *use_receive_zmq_option*. You could also set this property directly in the handler file (as in this sample). We assume the screenoption widget is called *screen_options* in designer:

```
#####
# **** INITIALIZE **** #
#####
# widgets allows access to widgets from the QtVCP files
# at this point the widgets and hal pins are not instantiated
def __init__(self, halcomp, widgets, paths):
    # directly select ZMQ message receiving
    self.w.screen_options.setProperty('use_receive_zmq_option', True)
```

This allows an external program to call functions in the handler file. Let's add a specific function for testing. You will need to run linuxcnc from a terminal to see the printed text.

```
#####
# general functions #
#####
def test_zmq_function(self, arg1, arg2):
    print('zmq test function called:', arg1, arg2)
```

Here is a sample program to call a function. It alternates between two data sets every second. Run this in a separate terminal from linuxcnc to see the sent messages.

```
#!/usr/bin/env python3
from time import sleep

import zmq
import json
```

```

context = zmq.Context()
socket = context.socket(zmq.PUB)
socket.bind("tcp://127.0.0.1:5690")
topic = b'QtVCP'

# prebuild message 1
# makes a dict of function to call plus any arguments
x = {
    "FUNCTION": "test_zmq_function",
    "ARGS": [True,200]
}
# convert to json object
m1 = json.dumps(x)

# prebuild message 2
x = {
    "FUNCTION": "test_zmq_function",
    "ARGS": [False,0],
}
# convert to json object
m2 = json.dumps(x)

if __name__ == '__main__':
    while True:
        print('send message 1')
        socket.send_multipart([topic, bytes((m1).encode('utf-8'))])
        sleep(ms(1000))

        print('send message 2')
        socket.send_multipart([topic, bytes((m2).encode('utf-8'))])
        sleep(ms(1000))

```

Note the line `x = {"FUNCTION": "test_zmq_function", "ARGS": [True,200]}` sets the function to call and the arguments to send to that function.

you will need to know the signature of the function you wish to call.

Also note that the message is converted to a json object.

This is because ZMQ sends byte messages not python objects.

json converts python to bytes and will be converted back when received.

10.4.11.14 external control with ZMQ messaging writing

You also may want to communicate with a separate program from the screen.

QtVCP can automatically set up ZMQ messaging to send and/or receive remote messages.

It uses ZMQ's publish/subscribe pattern of messages.

As always consider security before letting programs interface through messaging.

In the screenoptions widget, you can select the property `use_send_zmq_message`

You could also set this property directly in the handler file (as in this sample).

We assume the screenoption widget is called `screen_options` in designer:

```

#####
# **** INITIALIZE **** #
#####
# widgets allows access to widgets from the QtVCP files
# at this point the widgets and hal pins are not instantiated
def __init__(self, halcomp, widgets, paths):
    # directly select ZMQ message sending
    self.w.screen_options.setProperty('use_send_zmq_option', True)

```

This allows sending messages to a separate program.
 The message sent will depend on what the external program is expecting.
 Let's add a specific function for testing.
 You will need to run linuxcnc from a terminal to see the printed text.
 We assume the screenoption widget is called *screen_options* in designer.
 You need to add something to call this function, such as a button click.

```
#####
# general functions #
#####
def send_zmq_message(self):
    # This could be any python object json can convert
    message = {"name": "John", "age": 30}
    self.w.screen_options.send_zmq_message(message)
```

Here is a sample program that will receive the message and print it to the terminal.

```
import zmq
import json

# ZeroMQ Context
context = zmq.Context()

# Define the socket using the "Context"
sock = context.socket(zmq.SUB)

# Define subscription and messages with topic to accept.
topic = "" # all topics
sock.setsockopt(zmq.SUBSCRIBE, topic)
sock.connect("tcp://127.0.0.1:5690")

while True:
    topic, message = sock.recv_multipart()
    print('{} sent message:{}'.format(topic, json.loads(message)))
```

10.4.11.15 Sending messages to status bar or desktop notify dialogs

There several ways to report information to the user.
 A statusbar is used for short information to show the user.

+ Not all screens have a statusbar. This is how you would use it.
timeout is seconds. *statusbar* is the designer set name.

```
self.w.statusbar.showMessage(message, timeout * 1000)
```

You can use the STATUS Library to send a message to the notify Library
 if it is enabled (usually set in screen options widget).
 This will send the message to the statusbar and the desktop notify dialog.
 The messages are also recorded until the user erases them using controls.
 The users can recall any recorded messages.

There are several options. TEMPARAY will show the message for a short time only.


```
STATUS.TEMPORARY_MESSAGE
STATUS.OPERATOR_ERROR
STATUS.OPERATOR_TEXT
STATUS.NML_ERROR
STATUS.NML_TEXT
```

Here is an example of sending an operator message:

```
STATUS.emit('error', STATUS.OPERATOR_ERROR, 'message')
```

You can send messages thru linuxcnc's operator message functions. This are usually caught by the notify system, so are equal to above. They would be printed to the terminal as well.

```
ACTION.SET_DISPLAY_MESSAGE('MESSAGE')
ACTION.SET_ERROR_MESSAGE('MESSAGE')
```

10.4.11.16 Catch Focus Changes

Focus is used to direct user action such as keyboard entry to the proper widget. We may want to know what widget currently has focus:

```
fwidget = QtWidgets.QApplication.focusWidget()
if fwidget is not None:
    print("focus widget class: {} name: {}".format(fwidget, fwidget.objectName()))
```

Or we may want to know what widget has focus right when focus changes:

```
# at this point:
# the widgets are instantiated.
# the HAL pins are built but HAL is not set ready
def initialized__(self):
    QtWidgets.QApplication.instance().event_filter.focusIn.connect(self.focusInChanged)

#####
# general functions #
#####

def focusInChanged(self, widget):
    if isinstance(widget.parent(), type(self.w.gcode_editor.editor)):
        print('G-code Editor')
    elif isinstance(widget, type(self.w.gcodegraphics)):
        print('G-code Display')
    elif isinstance(widget.parent(), type(self.w.mdihistory)):
        print('MDI History')
```

Notice we sometimes compare to *widget*, sometimes *widget.parent()*.

This is because some QtVCP widgets are built from multiple sub-widgets and the sub-widgets actually get the focus; so we need to check the parent of those sub-widgets.

Other times the main widget is what gets the focus; ie G-code display widget can be set to accept focus and in that case there are no sub-widgets in it so comparing to the *widget.parent()* would get you the container that holds the G-code widget.

10.4.12 Qtvcp Development

The intention of QtVCP is to supply an infrastructure to support screen and VCP panel building for linuxcnc. By supplying a diverse widget set and supporting custom coding, QtVCP hopes that development energy will be expended in one toolkit rather than continuous re-invention. By using the same toolkit across many screens/panels users should have an easier time customizing/creating and developers should find it easier to help trouble shoot with less effort.

10.4.12.1 Overview

QtVCP uses a QT Designer built .ui file and a python handler file to load and control a screen/panel that displays QT widgets to control linuxcnc's motion controller or HAL pins. There are builtin screens and panels, easily loaded by a user or A user can build/modify one of their own. QtVCP uses libraries and custom widgets to hide some of the complexity of interfacing to linuxcnc. By using QtVCP's library rather than linuxcnc's, We can mitigate minor linuxcnc code changes.

10.4.12.2 Builtin locations

Builtin screens and panels are stored in separate folders.

- screens in share/qtvcv/screens
- panels in share/qtvcv/panels
- Stock images are in share/qtvcv/images

screens and panels are sorted by their folder name, which is also the name used to load them. Inside the folder would be the .ui file, the handler file and possibly the .qss theme file.

10.4.12.3 QtVCP startup to shutdown

QtVCP source is located in src/emc/usr_intf/qtvcv

- When QtVCP first starts, it must decide if this object is a screen or a panel. Then it searches for and collects information about paths of required files and useful folders.
 - Next, it builds the HAL component, loads the window instance, adds handler extensions and installs an event filter.
 - Now the window/widgets are instantiated, the HAL pins are built, which also initiates the `_init_hal()` function of the widgets.
 - The handler function `initialized_()` is called and then the STATUS library is forced to update. Hal component is set ready at this point.
 - A variety of optional switch arguments are set, including calling a POSTGUI HAL file (if a screen). Terminate signals are trapped and QtVCP now polls for events.
 - Finally when QtVCP is asked to shutdown, it calls shutdown functions in the handler file, shuts down STATUS monitoring and kills the HAL component.
-

10.4.12.4 Path information

When QtVCP loads it collects Path information: This is available in the handler file's `init()` function as *path*

- IMAGEDIR = Path of builtin images
- SCREENDIR = Path of builtin motion controller screens
- PANELDIR = Path of builtin accessory panels
- WORKINGDIR = Path of where QtVCP was launched from
- CONFIGPATH = Path of the launched configuration
- BASEDIR = general path, used to derive all paths
- BASENAME = Generic name used to derive all paths
- LIBDIR = Path of QtVCP's python library
- HANDLER = Path of handler file
- XML = Path of .ui file
- DOMAIN = Path of translation
- IS_SCREEN = screen/panel switch

10.4.12.5 Idiosyncrasies

These try to cover non-obvious situations.

Linuxcnc's error code collecting can only be read from one place. When read, it is *consumed* - no other object can read it. In QtVCP screens, it is recommended to use the ScreenOptions widget to set up error reading. They are then sent to other objects via STATUS signals.

Linuxcnc has no internal record of jog rate - you must specify it at the time of jogging. QtVCP uses the STATUS library to keep track of the latest linear and angular jog rate. It is always specified in machine units per minute, so must be converted when in non-machine units mode. So if your machine is imperial based but you are in metric mode changes to jog rate sent to ACTION functions must be converted to imperial. at the same time if the machine is metric based and you are in imperial mode, changes to jog rate must be sent to ACTION functions in metric units. Angular jog rates the units don't change in metric/imperial mode so you can send them to ACTION functions without conversion. While you are free to ignore this jogging record while building screens, Anyone modifying your screen and using the builtin jog rate widgets would not get the desired results. The ACTION library's `DO_JOG` function, gets it's jog rate from the STATUS library.

Keybinding is always a difficult-to-get-right-in-all-cases affair. Custom keybinding functions are to be defined in the handler file. Most importantly widgets that require regular key input and not jogging, should be checked for in the `processed_key_event__` function.

Some QtVCP widget use the preference file to record important information. This requires The preference file to be set up early in the widget initialization process. The easiest way to do this is to use the ScreenOptions widget.

QtVCP looks for and calls the `_hal_init` function, when the widget is first loaded. It is not called when using Designer editor. After this function is called the widget has access to some special variables:

```
self.HAL_GCOMP = the HAL component instance
self.HAL_NAME = This widgets name as a string
self.QT_OBJECT_ = This widgets pyQt object instance
self.QTVCP_INSTANCE_ = The very toplevel Parent Of the screen
self.PATHS_ = The instance of QtVCP's path library
self.PREFS_ = the instance of an optional preference file
self.SETTINGS_ = the Qsettings object
```

When making a custom widget, import and sub class the *_HalWidgetBase* class for this behavior:

Dialogs (AKA pop up windows) are best loaded with the screenoptions widget, but they can be placed on the screen in Designer. It doesn't matter where on the layout but to make them hidden, cycle the *state* property to true then false.

By default, if there is a preference file, the dialogs will remember their last size/placement. It is possible to override this so they open in the same location each time.

While it is possible to set styles in designer, it is more convenient to change them later if they are all set in a separate .qss file. The file should be put in the same location as the handler file.

Chapter 11

User Interfaces

11.1 Panelui

11.1.1 Introduction

Panelui is a userspace component to interface buttons to linuxcnc or HAL. It decodes MESA 7I73 style key-scan codes and calls the appropriate routine. It gets input from a realtime component - sampler. Sampler gets it's input from either the MESA 7i73 or `sim_matrix_kb` component. Panelui is configurable using an INI style text file to define button types, HAL pin types, and/or commands.

It can be extended using a Python based *handler* file to add functions. While actual input buttons are required to be momentary, Panelui will use this input to make toggle, radio, or momentary button output.

11.1.2 Loading Commands

The command used to load panelui (with optional -d debug switch):

```
loadusr -W panelui -d
```

This will initialize panelui, which will look for the INI file `panelui.ini` in the config folder or user folder. One can validate the INI file with this command:

```
loadusr pyui
```

This will read, try to correct, then save the `panelui.ini` file. It will print errors to the terminal if found. A typical HAL file will have these commands added:

```
# commands needed for panelui loading
#
# sampler is needed for panelui
# cfg= must always be u for panelui. depth sets the available buffer
loadrt sampler cfg=u depth=1025

#uncomment to validate the panelui INI file
#loadusr pyui

# -d = debug, -v = verbose debug
# -d will show you keypress identification and commands called
# -v is for developer info
```

```
loadusr -W panelui -d

# using simulated buttons instead of the MESA 7I73 card
# so we load the sim_matrix_kb component to convert HAL pins to keyscan codes
loadrt sim_matrix_kb

# connect the components together.
# sampler talks to panelui internally
net key-scan    sim-matrix-kb.0.out
net key-scan    sampler.0.pin.0

# add panelui components to a thread

addf sim-matrix-kb.0    servo-thread
addf sampler.0          servo-thread
```

11.1.3 panelui.ini file reference

Key words

- **KEY=** This is used to designate the key that the button responds to. It can be NONE or ROW number and column number eg R1C2. A row and column can only be used once.
- **OUTPUT=** This sets the Button's output type, eg S32, U32, FLOAT, BIT, NONE, COMMAND, ZMQ.
- **DEFAULT=** This sets the starting output of the group or button.
- **GROUP=** In radiobuttons, designates the group the button interacts with.
- **GROUP_OUTPUT=** sets the output the group pin will be, if this button is active.
- **STATUS_PIN=** If TRUE, a HAL pin will be added that reflects the current state of button.
- **TRUE_STATE=** sets the output the HAL pin will be, if button is TRUE.
- **FALSE_STATE=** sets the OUTPUT the HAL pin will be, if the button is FALSE.
- **TRUE_COMMAND=** sets the command and arguments to be called when the button is TRUE.
- **FALSE_COMMAND=** sets the command and arguments to be called when the button is FALSE.
- **TRUE_FUNCTION=** sets the ZMQ message function and arguments to be called when the button is TRUE.
- **FALSE_FUNCTION=** sets the ZMQ message function and arguments to be called when the button is FALSE.

HAL Prefix

```
[HAL_PREFIX]
NAME= Yourname
```

This allows one to change the prefix of the HAL pins from *panelui* to an arbitrary name.

ZMQ Messaging Setup

```
[ZMQ_SETUP]
TOPIC = 'QTVCP'
SOCKET = 'tcp://127.0.0.1:5690'
ENABLE = True
```

This sets up and enables ZMQ based messaging. TOPIC and SOCKET must match the listening program.

Radio Buttons Radiobuttons allow only one button in the group to be active at a time. Each group has its own output pin, separate from each button in the group. Radio button definitions start with the text *RADIO_BUTTON* inside single brackets.

```
[RADIO_BUTTONS]
# The double bracket section(s) define the group(s) of radio buttons.
# The group name must be unique and is case sensitive.
# Groups output is controlled by what button is active not directly by keycode.
# DEFAULT references a button in the group by name and is case sensitive.
[[group1_name]]
KEY = NONE
OUTPUT = FLOAT
DEFAULT = small
# The triple bracket sections define the buttons in this group.
# button names must be unique and are case sensitive.
# There must be at least two buttons in a group.
#
# This button, named 'small' is controller by the row 0 column 1 key.
# It will cause the group output to be .0001 when it is pressed.
# It has no output of it's own, but has a status
# pin which will follow it's current state.
# since this button is in a group, DEFAULT has no bearing.
# since OUTPUT in not 'COMMAND' _COMMAND entries are ignored.
[[[small]]]
KEY = R0C1
GROUP = group1_name
GROUP_OUTPUT = .0001
OUTPUT = NONE
STATUS_PIN = True
TRUE_STATE = TRUE
FALSE_STATE = FALSE
TRUE_COMMAND = NONE, NONE
FALSE_COMMAND = NONE, NONE
DEFAULT = false
# This button, named 'large' is controller by the row 0 column 2 key.
# It will cause the group output to be 1000 when it is pressed.
# It has a S32 output of it's own, will be 20 on true and 0 on false.
# It also has a status pin which will follow it's current state.
# since this button is in a group, DEFAULT has no bearing.
# since OUTPUT in not 'COMMAND' _COMMAND entries are ignored.
[[[large]]]
KEY = R0C2
GROUP = group1_name
GROUP_OUTPUT = 1000
OUTPUT = S32
STATUS_PIN = True
TRUE_STATE = 20
TRUE_COMMAND = NONE, NONE
FALSE_COMMAND = NONE, NONE
FALSE_STATE = 0
DEFAULT = false
```

Toggle Buttons Togglebuttons only change state on each press of the button. Toggle button definitions start with the text *TOGGLE_BUTTON* inside single brackets.

```
[TOGGLE_BUTTONS]
# Each button name inside double brackets, must be unique and is case sensitive.
# This button, named 'tool_change' is controller by the row 2 column 5 key.
# It has a BIT output, will output 1 on true state and 0 on false state.
```

```
# It also has a status pin which will follow it's current state.
# DEFAULT sets this to true when first initialized.
# The _COMMAND are not used since OUTPUT is not set to COMMAND but validation will
# add the lines regardless
[[tool_change]]
    KEY = R2C5
    OUTPUT = BIT
    TRUE_COMMAND = NONE, NONE
    FALSE_COMMAND = NONE, NONE
    STATUS_PIN = True
    DEFAULT = TRUE
    TRUE_STATE = 1
    FALSE_STATE = 0
```

Momentary Buttons Momentary buttons are true when pressed and false when released. Momentary button definitions start with the text *MOMENTARY_BUTTON* inside single brackets.

```
[MOMENTARY_BUTTONS]
# Each button name inside double brackets, must be unique and is case sensitive.
# This button, named 'spindle_rev' is controller by the row 2 column 3 key.
# It has a COMMAND output, so will use TRUE_COMMAND and FALSE_COMMAND.
# It also has a status pin which will follow it's current state.
# COMMANDs will have a command name and then any required arguments
# This TRUE_COMMAND calls an internal command to start the spindle in reverse at 200 rpm
# If the spindle is already started, it will increase the rpm.
# DEFAULT is not used with Momentary buttons.
# The _STATE are not used since OUTPUT is set to COMMAND but validation will
# add the lines regardless
[[spindle_rev]]
    KEY = R2C3
    OUTPUT = COMMAND
    TRUE_COMMAND = SPINDLE_REVERSE_INCREASE, 200
    FALSE_COMMAND = None, NONE
    STATUS_PIN = True
    DEFAULT = FALSE
    TRUE_STATE = 1
    FALSE_STATE = 0
```

11.1.4 Internal Command reference

There are a number of internal commands you may use.

home_selected

- required argument: axis number (int)

unhome_selected

- required argument: axis number (int)

spindle_forward_adjust

- optional argument: starting RPM (int) - default 100
- Description: If the spindle is stopped it will start in the forward direction. If it is already running it will increase or decrease the rpm depending on what direction the spindle is running in.

spindle_forward

- optional argument: starting RPM (int) - default 100

spindle_reverse

- optional argument: starting RPM (int) - default 100

spindle_reverse_adjust

- optional argument: starting RPM (int) - default 100
- Description: If the spindle is stopped it will start in the reverse direction. If it is already running it will increase or decrease the rpm depending on what direction the spindle is running in.

spindle_faster

- Description: increases spindle speed by 100 RPM

spindle_slower

- Description: decreases spindle speed by 100 RPM, until RPM is 100

set_linear_jog_velocity

- required argument: velocity in inches per minute (float)
- description: sets the jog velocity on axis 0,1,2,6,7,8 (X,Y,Z,U,V,W)

setAngular_jog_velocity

- required argument: velocity in degrees per minute (float)
- description: sets the jog velocity on axis 3,4,5 (A,B,C)

continuous_jog

- required arguments: axis number (int), direction (int)

incremental_jog

- required arguments: axis number (int), direction (int), distance (float)

quill_up

- optional arguments: machine Z axis absolute position (float)
- Description: Move Z axis to the given machine position

feed_hold

- required argument: state (bool 0 or 1)

feed_override

- required argument: rate (float)

rapid_override

- required argument: rate (float 0-1)

spindle_override

- required argument: rate (float)

max_velocity

- required argument: rate (float)

optional_stop

- required argument: state (bool 0 or 1)

block_delete

- required argument: state (bool 0 or 1)

single_block

- required argument: state (bool 0 or 1)

smart_cycle_start

- Description: If idle, starts G-code program, if paused runs one line.

re_start line

- required argument: line number (int)

mdi_and_return

- required argument: G-code command(s)
- Description: records the current mode, calls commands and then returns to mode.

mdi

- required argument: G-code command(s)
- Description: sets mode to MDI, calls commands.

11.1.5 ZMQ Messages

panelui can send ZMQ based messages on button presses.

In this way panelui can interact will other programs such as QtVCP screens.

```
[TOGGLE_BUTTONS]
[[zmq_test]]
KEY = R2C3
OUTPUT = ZMQ
TRUE_FUNCTION = ZMQ_BUTTON, 200
FALSE_FUNCTION = ZMQ_BUTTON, 0
STATUS_PIN = False
DEFAULT = FALSE
TRUE_STATE = 1
FALSE_STATE = 0
```

Here is a sample program that will receive the message and print it to the terminal.

```
import zmq
import json

# ZeroMQ Context
context = zmq.Context()

# Define the socket using the "Context"
sock = context.socket(zmq.SUB)

# Define subscription and messages with topic to accept.
topic = "" # all topics
sock.setsockopt(zmq.SUBSCRIBE, topic)
sock.connect("tcp://127.0.0.1:5690")

while True:
    topic, message = sock.recv_multipart()
    print('{} sent message:{}'.format(topic, json.loads(message)))
```

11.1.6 Handler File Extension

A special file can be used to add custom python code that will be available as commands. `panelui_handler.py` must be written in python and be placed in the configuration folder. If panelui finds a file there it will add it's function calls to the available commands. Here is an example of a handler file that adds two functions - `hello_world` and `cycle_mode`:

```
# standard handler call - This will always be required
def get_handlers(linuxcnc_stat, linuxcnc_cmd, commands, master):
    return [HandlerClass(linuxcnc_stat, linuxcnc_cmd, commands, master)]

# Also required - handler class
class HandlerClass:

    # This will be pretty standard to gain access to everything
    # linuxcnc_stat: is the python status instance of linuxcnc
    # linuxcnc_cmd: is the python command instance of linuxcnc
    # commands: is the command instance so one can call the internal routines
    # master: give access to the master functions/data

    def __init__(self, linuxcnc_stat, linuxcnc_cmd, commands, master):
        self.parent = commands
        self.current_mode = 0

    # command functions are expected to have this layout:
    # def some_name(self, widget_instance, arguments from widget):
    # widget_instance gives access to the calling widget's function/data
    # arguments can be a list of arguments, a single argument, or None
    # depending on what was given in panelui's INI file.
    def hello_world(self, wname, m):
        # print to terminal so we know it worked
        print('\nHello world\n')
        print(m) # print the argument(s)
        print(wname.metadata) # Print the calling widgets internal metadata (from config ↔
                           file)

        # call a mdi command to print a msg in linuxcnc
        # This requires linuxcnc to be homed, but does not check for that.
        # parent commands expect a widget_instance - None is substituted
```

```

        self.parent.mdi(None, '(MSG, Hello Linuxcnc World!)')

# Each call to this function will cycle the mode of linuxcnc
def cycle_mode(self, wname, m):
    if self.current_mode == 0:
        self.current_mode = 1
        self.parent.set_mdi_mode()
    elif self.current_mode == 1:
        self.current_mode = 2
        self.parent.set_auto_mode()
    else:
        self.current_mode = 0
        self.parent.set_manual_mode()
    print(self.current_mode)

# Boiler code, often required
def __getitem__(self, item):
    return getattr(self, item)
def __setitem__(self, item, value):
    return setattr(self, item, value)

```

11.2 HAL User Interface

11.2.1 Introduction

Halui is a HAL based user interface for LinuxCNC, it connects HAL pins to NML commands. Most of the functionality (buttons, indicators etc.) that is provided by a traditional GUI (AXIS, GMoccapy, QtDragon, etc.), is provided by HAL pins in Halui.

The easiest way to add halui is to add the following to the [HAL] section of the INI file:

```

[HAL]
HALUI = halui

```

An alternate way to invoke it (specially if you generate the configuration with StepConf) is to include the following in your custom.hal file.

Make sure you use the correct path to your INI file.

```
loadusr halui -ini /path/to/inifile.ini
```

11.2.2 MDI

Sometimes the user wants to add more complicated tasks to be performed by the activation of a HAL pin. This is possible by adding MDI commands to the INI file in the [HALUI] section. Example:

```

[HALUI]
MDI_COMMAND = G0 X0
MDI_COMMAND = G0 G53 Z0
MDI_COMMAND = G28
MDI_COMMAND = o<mysub>call
...

```

When halui starts it will read the MDI_COMMAND fields in the INI and export pins from 00 to the number of MDI_COMMAND 's found in the INI, up to a maximum of 64 commands. These pins can be connected like any HAL pins. A common method is to use buttons provided by virtual control panels. Example HAL file connections:

```

net quill-up      <= pyvcp.quillup
net quill-up      => halui.mdi-command-00
net reference-pos <= pyvcp.referencepos
net reference-pos => halui.mdi-command-01
net call-mysub    <= pyvcp.callmysub
net call-mysub    => halui.mdi-command-02

```

These nets connect the `halui.mdi-command-NN` pins provided by `halui`:

```

$ halcmd show pin halui.mdi
Component Pins:
Owner  Type  Dir Value  Name
10 bit  IN  FALSE  halui.mdi-command-00 <== quill-up
10 bit  IN  FALSE  halui.mdi-command-01 <== reference-pos
10 bit  IN  FALSE  halui.mdi-command-02 <== call-mysub
...

```

When a `halui` MDI pin is set (pulsed) true, `halui` will send the MDI command defined in the INI. This will not always succeed depending on the current operating mode (e.g., while in AUTO `halui` can't successfully send MDI commands).

11.2.3 Example Configuration

An example sim config (`configs/sim/axis/halui_pyvcp/halui.ini`) is included in the distribution.

11.2.4 Halui Pin Reference

All `halui` pins are also documented in the `halui` man page:

```
$ man halui
```

11.2.4.1 Abort

- `halui.abort` (bit, in) - pin to send an abort message (clears out most errors)

11.2.4.2 E-Stop

- `halui.estop.activate` (bit, in) - pin for requesting E-Stop
- `halui.estop.is-activated` (bit, out) - indicates E-stop reset
- `halui.estop.reset` (bit, in) - pin for requesting E-Stop reset

11.2.4.3 Feed Override

- `halui.feed-override.count-enable` (bit, in) - must be true for *counts* or *direct-value* to work.
- `halui.feed-override.counts` (s32, in) - `counts * scale = FO percentage`. Can be used with an encoder or *direct-value*.
- `halui.feed-override.decrease` (bit, in) - pin for decreasing the FO (`-=scale`)
- `halui.feed-override.increase` (bit, in) - pin for increasing the FO (`+=scale`)

- *halui.feed-override.reset* (bit, in) - pin for resetting the FO (scale=1.0)
- *halui.feed-override.direct-value* (bit, in) - false when using encoder to change counts, true when setting counts directly.
- *halui.feed-override.scale* (float, in) - pin for setting the scale for increase and decrease of *feed-override*.
- *halui.feed-override.value* (float, out) - current FO value

11.2.4.4 Mist

- *halui.mist.is-on* (bit, out) - indicates mist is on
- *halui.mist.off* (bit, in) - pin for requesting mist off
- *halui.mist.on* (bit, in) - pin for requesting mist on

11.2.4.5 Flood

- *halui.flood.is-on* (bit, out) - indicates flood is on
- *halui.flood.off* (bit, in) - pin for requesting flood off
- *halui.flood.on* (bit, in) - pin for requesting flood on

11.2.4.6 Homing

- *halui.home-all* (bit, in) - pin for requesting all axis to home. This pin will only be there if HOME_SEQUENCE is set in the INI file.

11.2.4.7 Lube

- *halui.lube.is-on* (bit, out) - indicates lube is on
- *halui.lube.off* (bit, in) - pin for requesting lube off
- *halui.lube.on* (bit, in) - pin for requesting lube on

11.2.4.8 Machine

- *halui.machine.units-per-mm* (float out) - pin for machine units-per-mm (inch:1/25.4, mm:1) according to inifile setting: [TRAJ]LINEAR_UNITS
 - *halui.machine.is-on* (bit, out) - indicates machine on
 - *halui.machine.off* (bit, in) - pin for requesting machine off
 - *halui.machine.on* (bit, in) - pin for requesting machine on
-

11.2.4.9 Max Velocity

The maximum linear velocity can be adjusted from 0 to the MAX_VELOCITY that is set in the [TRAJ] section of the INI file.

- *halui.max-velocity.count-enable* (bit, in) - must be true for *counts* or *direct-value* to work.
- *halui.max-velocity.counts* (s32, in) - counts * scale = MV percentage. Can be used with an encoder or *direct-value*.
- *halui.max-velocity.direct-value* (bit, in) - false when using encoder to change counts, true when setting counts directly.
- *halui.max-velocity.decrease* (bit, in) - pin for decreasing max velocity
- *halui.max-velocity.increase* (bit, in) - pin for increasing max velocity
- *halui.max-velocity.scale* (float, in) - the amount applied to the current maximum velocity with each transition from off to on of the increase or decrease pin in machine units per second.
- *halui.max-velocity.value* (float, out) - is the maximum linear velocity in machine units per second.

11.2.4.10 MDI

- *halui.mdi-command-<nn>* (bit, in) - halui will try to send the MDI command defined in the INI. <nn> is a two digit number starting at 00.
If the command succeeds then it will place LinuxCNC in the MDI mode and then back to Manual mode.
If no [HALUI]MDI_COMMAND variables are set in the ini file, no *halui.mdi-command-<nn>* pins will be exported by halui.

11.2.4.11 Joint

N = joint number (0 ... num_joints-1)

Example:

- *halui.joint.N.select* (bit in) - pin for selecting joint N
- *halui.joint.N.is-selected* (bit out) - status pin that joint N is selected
- *halui.joint.N.has-fault* (bit out) - status pin telling that joint N has a fault
- *halui.joint.N.home* (bit in) - pin for homing joint N
- *halui.joint.N.is-homed* (bit out) - status pin telling that joint N is homed
- *halui.joint.N.on-hard-max-limit* (bit out) - status pin telling that joint N is on the positive hardware limit
- *halui.joint.N.on-hard-min-limit* (bit out) - status pin telling that joint N is on the negative hardware limit
- *halui.joint.N.on-soft-max-limit* (bit out) - status pin telling that joint N is on the positive software limit
- *halui.joint.N.on-soft-min-limit* (bit out) - status pin telling that joint N is on the negative software limit
- *halui.joint.N.override-limits* (bit out) - status pin telling that joint N's limits are temporarily overridden

- *halui.joint.N.unhome* (bit in) - pin for unhoming joint N
- *halui.joint.selected* (u32 out) - selected joint number (0 ... num_joints-1)
- *halui.joint.selected.has-fault* (bit out) - status pin selected joint is faulted
- *halui.joint.selected.home* (bit in) - pin for homing the selected joint
- *halui.joint.selected.is-homed* (bit out) - status pin telling that the selected joint is homed
- *halui.joint.selected.on-hard-max-limit* (bit out) - status pin telling that the selected joint is on the positive hardware limit
- *halui.joint.selected.on-hard-min-limit* (bit out) - status pin telling that the selected joint is on the negative hardware limit
- *halui.joint.selected.on-soft-max-limit* (bit out) - status pin telling that the selected joint is on the positive software limit
- *halui.joint.selected.on-soft-min-limit* (bit out) - status pin telling that the selected joint is on the negative software limit
- *halui.joint.selected.override-limits* (bit out) - status pin telling that the selected joint's limits are temporarily overridden
- *halui.joint.selected.unhome* (bit in) - pin for unhoming the selected joint

11.2.4.12 Joint Jogging

N = joint number (0 ... num_joints-1)

- *halui.joint.jog-deadband* (float in) - pin for setting jog analog deadband (jog analog inputs smaller/slower than this - in absolute value - are ignored)
- *halui.joint.jog-speed* (float in) - pin for setting jog speed for plus/minus jogging.
- *halui.joint.N.analog* (float in) - pin for jogging the joint N using a float value (e.g. joy-stick). The value, typically set between 0.0 and ± 1.0 , is used as a jog-speed multiplier.
- *halui.joint.N.increment* (float in) - pin for setting the jog increment for joint N when using increment-plus/minus
- *halui.joint.N.increment-minus* (bit in) - a rising edge will will make joint N jog in the negative direction by the increment amount
- *halui.joint.N.increment-plus* (bit in) - a rising edge will will make joint N jog in the positive direction by the increment amount
- *halui.joint.N.minus* (bit in) - pin for jogging joint N in negative direction at the *halui.joint.jog-speed* velocity
- *halui.joint.N.plus* (bit in) - pin for jogging joint N in positive direction at the *halui.joint.jog-speed* velocity
- *halui.joint.selected.increment* (float in) - pin for setting the jog increment for the selected joint when using increment-plus/minus
- *halui.joint.selected.increment-minus* (bit in) - a rising edge will will make the selected joint jog in the negative direction by the increment amount
- *halui.joint.selected.increment-plus* (bit in) - a rising edge will will make the selected joint jog in the positive direction by the increment amount

- *halui.joint.selected.minus* (bit in) - pin for jogging the selected joint in negative direction at the *halui.joint.jog-speed* velocity
- *halui.joint.selected.plus* (bit in) - pin for jogging the selected joint in positive direction at the *halui.joint.jog-speed* velocity

11.2.4.13 axis

L = axis letter (xyzabcuvw)

- *halui.axis.L.select* (bit) - pin for selecting axis by letter
- *halui.axis.L.is-selected* (bit out) - status pin that axis L is selected
- *halui.axis.L.pos-commanded* (float out) - Commanded axis position in machine coordinates
- *halui.axis.L.pos-feedback* (float out) - Feedback axis position in machine coordinates
- *halui.axis.L.pos-relative* (float out) - Feedback axis position in relative coordinates

11.2.4.14 Axis Jogging

L = axis letter (xyzabcuvw)

- *halui.axis.jog-deadband* (float in) - pin for setting jog analog deadband (jog analog inputs smaller/lower than this (in absolute value) are ignored)
- *halui.axis.jog-speed* (float in) - pin for setting jog speed for plus/minus jogging.
- *halui.axis.L.analog* (float in) - pin for jogging the axis L using an float value (e.g. joystick). The value, typically set between 0.0 and ± 1.0 , is used as a jog-speed multiplier.
- *halui.axis.L.increment* (float in) - pin for setting the jog increment for axis L when using increment-plus/minus
- *halui.axis.L.increment-minus* (bit in) - a rising edge will will make axis L jog in the negative direction by the increment amount
- *halui.axis.L.increment-plus* (bit in) - a rising edge will will make axis L jog in the positive direction by the increment amount
- *halui.axis.L.minus* (bit in) - pin for jogging axis L in negative direction at the *halui.axis.jog-speed* velocity
- *halui.axis.L.plus* (bit in) - pin for jogging axis L in positive direction at the *halui.axis.jog-speed* velocity
- *halui.axis.selected* (u32 out) - selected axis (by index: 0:x 1:y 2:z 3:a 4:b 5:cr 6:u 7:v 8:w)
- *halui.axis.selected.increment* (float in) - pin for setting the jog increment for the selected axis when using increment-plus/minus
- *halui.axis.selected.increment-minus* (bit in) - a rising edge will will make the selected axis jog in the negative direction by the increment amount
- *halui.axis.selected.increment-plus* (bit in) - a rising edge will will make the selected axis jog in the positive direction by the increment amount
- *halui.axis.selected.minus* (bit in) - pin for jogging the selected axis in negative direction at the *halui.axis.jog-speed* velocity
- *halui.axis.selected.plus* (pin in) - for jogging the selected axis bit in in positive direction at the *halui.axis.jog-speed* velocity

11.2.4.15 Mode

- *halui.mode.auto* (bit, in) - pin for requesting auto mode
- *halui.mode.is-auto* (bit, out) - indicates auto mode is on
- *halui.mode.is-joint* (bit, out) - indicates joint by joint jog mode is on
- *halui.mode.is-manual* (bit, out) - indicates manual mode is on
- *halui.mode.is-mdi* (bit, out) - indicates MDI mode is on
- *halui.mode.is-teleop* (bit, out) - indicates coordinated jog mode is on
- *halui.mode.joint* (bit, in) - pin for requesting joint by joint jog mode
- *halui.mode.manual* (bit, in) - pin for requesting manual mode
- *halui.mode.mdi* (bit, in) - pin for requesting MDI mode
- *halui.mode.teleop* (bit, in) - pin for requesting coordinated jog mode

11.2.4.16 Program

- *halui.program.block-delete.is-on* (bit, out) - status pin telling that block delete is on
- *halui.program.block-delete.off* (bit, in) - pin for requesting that block delete is off
- *halui.program.block-delete.on* (bit, in) - pin for requesting that block delete is on
- *halui.program.is-idle* (bit, out) - status pin telling that no program is running
- *halui.program.is-paused* (bit, out) - status pin telling that a program is paused
- *halui.program.is-running* (bit, out) - status pin telling that a program is running
- *halui.program.optional-stop.is-on* (bit, out) - status pin telling that the optional stop is on
- *halui.program.optional-stop.off* (bit, in) - pin requesting that the optional stop is off
- *halui.program.optional-stop.on* (bit, in) - pin requesting that the optional stop is on
- *halui.program.pause* (bit, in) - pin for pausing a program
- *halui.program.resume* (bit, in) - pin for resuming a paused program
- *halui.program.run* (bit, in) - pin for running a program
- *halui.program.step* (bit, in) - pin for stepping in a program
- *halui.program.stop* (bit, in) - pin for stopping a program

11.2.4.17 Rapid Override

- *halui.rapid-override.count-enable* (bit in (default: TRUE)) - When TRUE, modify Rapid Override when counts changes.
 - *halui.rapid-override.counts* (s32 in) - counts X scale = Rapid Override percentage. Can be used with an encoder or *direct-value*.
 - *halui.rapid-override.decrease* (bit in) - pin for decreasing the Rapid Override (=-scale)
 - *halui.rapid-override.direct-value* (bit in) - pin to enable direct value Rapid Override input
-

- *halui.rapid-override.increase* (bit in) - pin for increasing the Rapid Override ($+=scale$)
- *halui.rapid-override.scale* (float in) - pin for setting the scale on changing the Rapid Override
- *halui.rapid-override.value* (float out) - current Rapid Override value
- *halui.rapid-override.reset* (bit, in) - pin for resetting the Rapid Override value ($scale=1.0$)

11.2.4.18 Spindle Override

- *halui.spindle.N.override.count-enable* (bit, in) - must be true for *counts* or *direct-value* to work.
- *halui.spindle.N.override.counts* (s32, in) - $counts * scale = SO$ percentage. Can be used with an encoder or *direct-value*.
- *halui.spindle.N.override.decrease* (bit, in) - pin for decreasing the SO ($-=scale$)
- *halui.spindle.N.override.direct-value* (bit, in) - false when using encoder to change counts, true when setting counts directly.
- *halui.spindle.N.override.increase* (bit, in) - pin for increasing the SO ($+=scale$)
- *halui.spindle.N.override.scale* (float, in) - pin for setting the scale on changing the SO
- *halui.spindle.N.override.value* (float, out) - current SO value
- *halui.spindle.N.override.reset* (bit, in) - pin for resetting the SO value ($scale=1.0$)

11.2.4.19 Spindle

- *halui.spindle.N.brake-is-on* (bit, out) - indicates brake is on
 - *halui.spindle.N.brake-off* (bit, in) - pin for deactivating spindle/brake
 - *halui.spindle.N.brake-on* (bit, in) - pin for activating spindle-brake
 - *halui.spindle.N.decrease* (bit, in) - decreases spindle speed
 - *halui.spindle.N.forward* (bit, in) - starts the spindle with CW motion
 - *halui.spindle.N.increase* (bit, in) - increases spindle speed
 - *halui.spindle.N.is-on* (bit, out) - indicates spindle is on (either direction)
 - *halui.spindle.N.reverse* (bit, in) - starts the spindle with a CCW motion
 - *halui.spindle.N.runs-backward* (bit, out) - indicates spindle is on, and in reverse
 - *halui.spindle.N.runs-forward* (bit, out) - indicates spindle is on, and in forward
 - *halui.spindle.N.start* (bit, in) - starts the spindle
 - *halui.spindle.N.stop* (bit, in) - stops the spindle
-

11.2.4.20 Tool

- *halui.tool.length-offset.a* (float out) - current applied tool length offset for the A axis
- *halui.tool.length-offset.b* (float out) - current applied tool length offset for the B axis
- *halui.tool.length-offset.c* (float out) - current applied tool length offset for the C axis
- *halui.tool.length-offset.u* (float out) - current applied tool length offset for the U axis
- *halui.tool.length-offset.v* (float out) - current applied tool length offset for the V axis
- *halui.tool.length-offset.w* (float out) - current applied tool length offset for the W axis
- *halui.tool.length-offset.x* (float out) - current applied tool length offset for the X axis
- *halui.tool.length-offset.y* (float out) - current applied tool length offset for the Y axis
- *halui.tool.length-offset.z* (float out) - current applied tool length offset for the Z axis
- *halui.tool.diameter* (float out) - Current tool diameter, or 0 if no tool is loaded.
- *halui.tool.number* (u32, out) - indicates current selected tool

11.3 Halui Examples

For any Halui examples to work you need to add the following line to the [HAL] section of the ini file.

```
HALUI = halui
```

11.3.1 Remote Start

To connect a remote program start button to LinuxCNC you use the *halui.program.run* pin and the *halui.mode.auto* pin. You have to insure that it is OK to run first by using the *halui.mode.is-auto* pin. You do this with an *and2* component. The following figure shows how this is done. When the Remote Run Button is pressed it is connected to both *halui.mode.auto* and *and2.0.in0*. If it is OK for auto mode the pin *halui.mode.is-auto* will be on. If both the inputs to the *and2.0* component are on the *and2.0.out* will be on and this will start the program.



Figure 11.1: Remote Start Example

The hal commands needed to accomplish the above are:

```

net program-start-btn halui.mode.auto and2.0.in0 <= <your input pin>
net program-run-ok and2.0.in1 <= halui.mode.is-auto
net remote-program-run halui.program.run <= and2.0.out

```

Notice on line one that there are two reader pins, this can also be split up to two lines like this:

```

net program-start-btn halui.mode.auto <= <your input pin>
net program-start-btn and2.0.in0

```

11.3.2 Pause & Resume

This example was developed to allow LinuxCNC to move a rotary axis on a signal from an external machine. The coordination between the two systems will be provided by two Halui components:

- halui.program.is-paused
- halui.program.resume

In your customized hal file, add the following two lines that will be connected to your I/O to turn on the program pause or to resume when the external system wants LinuxCNC to continue.

```

net ispaused halui.program.is paused => "your output pin"
net resume halui.program.resume <= "your input pin"

```

Your input and output pins are connected to the pins wired to the other controller. They may be parallel port pins or any other I/O pins that you have access to.

This system works in the following way. When an M0 is encountered in your G-code, the halui.program.is-paused signal goes true. This turns on your output pin so that the external controller knows that LinuxCNC is paused.

To resume the LinuxCNC G-code program, when the external controller is ready it will make its output true. This will signal LinuxCNC that it should resume executing G-code.

Difficulties in timing

- The "resume" input return signal should not be longer than the time required to get the G-code running again.
- The "is-paused" output should no longer be active by the time the "resume" signal ends.

These timing problems could be avoided by using ClassicLadder to activate the "is-paused" output via a monostable timer to deliver one narrow output pulse. The "resume" pulse could also be received via a monostable timer.

11.4 Vismach

Vismach is a set of Python functions that can be used to create and animate models of machines. Vismach displays the model in a 3D viewport and the model parts are animated as the values of associated HAL pins change.



The Vismach viewport view can be manipulated as follows:

- **zoom** by scroll wheel or right button drag
- **pan** by left button drag
- **rotate** by middle-button drag or shift-drag.

A Vismach model takes the form of a Python script and can use standard Python syntax. This means that there is more than one way to lay out the script, but in the examples given in this document I will use the simplest and most basic of them.

The basic sequence in creating the Vismach model is

- Create the HAL pins that control the motion
- Create the parts

- Define how they move
- Assemble into movement groups

11.4.1 Start the script

It is useful for testing to include the `#!/usr/bin/env python3` to allow the file to be run as a script. The first thing to do is to import the required libraries.

```
#!/usr/bin/env python3

from vismach import *
import hal
import math
import sys
```

11.4.2 Create the HAL pins.

Hal pins are created with the normal Python "hal" library, and are not specific to Vismach. Further details can be found in the [Creating Userspace Components in Python](#) section. A component should be created with a name that matches the script file name and then the HAL pins are added to that component. They will be referenced by their component handle and short name when used to animate the Vismach model.

```
c = hal.component("samplegui")
c.newpin("joint0", hal.HAL_FLOAT, hal.HAL_IN)
c.newpin("joint1", hal.HAL_FLOAT, hal.HAL_IN)
c.ready()
```

Will create HAL pins *samplegui.joint0* and *samplegui.joint1*. When loading the Vismach model with `loadusr -W samplegui` the `c.ready()` function tells loadusr it's ready.

11.4.3 Creating Parts

It is probably easiest to create geometry in a CAD package and **import** into the model script with the `AsciiSTL()` or `AsciiOBJ()` functions. Both functions can take one of two named arguments, either a filename or raw data

- `part = AsciiSTL(filename="path/to/file.stl")` + `part = AsciiSTL(data="solid part1 facet normal")` + `part = AsciiOBJ(filename="path/to/file.obj")` + `part = AsciiOBJ(data="v 0.123 0.234 0.345 1.0 ...")`

The parts will be created in the Vismach space in the same locations as they occupy in the STL or OBJ space. This means that it may be possible to assemble the model in the CAD package.

Alternatively parts can be created inside the model script from a range of **shape primitives**. Many shapes are created at the origin and need to be moved to the required location after creation.

- `cylinder = CylinderX(x1, r1, x2, r2)` + `cylinder = CylinderY(y1, r1, y2, r2)` + `cylinder = CylinderZ(z1, r1, z2, r2)`
Creates a (optionally tapered) cylinder on the given axis with the given radii at the given points on the axis.
- `sphere = Sphere(x, y, z, r)`
Creates a sphere of radius r at (x,y,z)

- `triangle = TriangleXY(x1, y1, x2, y2, x3, y3, z1, z2) + triangle = TriangleXZ(x1, z1, x2, z2, x3, z3, y1, y2) + triangle = TriangleYZ(y1, z1, y2, z2, y3, z3, x1, x2)`
Creates a triangular plate between planes defined by the last two values parallel to the specified plane, with vertices given by the three coordinate pairs.
- `arc = ArcX(x1, x2, r1, r2, a1, a2)`
Create an arc shape.
- `box = Box(x1, y1, z1, x2, y2, z2)`
Creates a rectangular prism with opposite corners at the specified positions and edges parallel to the XYZ axes.
- `box = BoxCentered(xw, yw, zw)`
Creates an `xw` by `yw` by `zw` box centred on the origin.
- `box = BoxCenteredXY(xw, yw, z)`
Creates a box of width `xw` / `yw` and height `z`.

Composite parts may be created by **assembling** these primitives either at creation time or subsequently using `Collection()`:

```
part1 = Collection([Sphere(100,100,100,50), CylinderX(100,40,150,30)])
part2 = Box(50,40,75,100,75,100)
part3 = Collection([part2, TriangleXY(10,10,20,10,15,20,100,101)])
part4 = Collection([part1, part2])
```

11.4.4 Moving Parts

Parts may need to be moved in the Vismach space to assemble the model. They may also need to be moved to create the animation as the animation rotation axis is created at the origin (but moves with the Part).

- `part1 = Translate([part1], x, y, z)`
Move `part1` the specified distances in `x`, `y` and `z`.
- `part1 = Rotate([part1], theta, x, y, z)`
Rotate the part by angle `theta` about an axis between the origin and `x`, `y`, `z`.

11.4.5 Animating Parts

To animate the model (controlled by the values of HAL pins) there are two functions *HalTranslate* and *HalRotate*. For parts to move inside an assembly they need to have their HAL motions defined before being assembled with the "Collection" command. The rotation axis and translation vector move with the part as it is moved by the vismach script during model assembly, or as it moves in response to the HAL pins as the model is animated.

- `part = HalTranslate([part], comp, "hal_pin", xs, ys, zs)`
The function arguments are:
 - first a *collection/part* which can be pre-created earlier in the script, or could be created at this point if preferred eg `part1 = HalTranslate([Box(....)], ...)`.
 - The *HAL component* is the next argument, ie the object returned by the `comp = hal.component(...)` command. After that is the name of the HAL in that will animate the motion, this needs to match an existing HAL pin that is part of the HAL component created earlier in the script.

- Then follow the *X, Y, Z scales*.
For a Cartesian machine created at 1:1 scale this would typically be 1,0,0 for a motion in the positive X direction.
However if the STL file happened to be in cm and the machine was in inches, this could be fixed at this point by using 0.3937 (1cm /2.54in) as the scale.
- `part = HalRotate([part], comp, "hal_pin", angle_scale, x, y, z)`
This command is similar in its operation to `HalTranslate` except that it is typically necessary to move the part to the origin first to define the axis.
- The *axis of rotation* is from the origin point to the point defined by (x,y,z).
When the part is moved back away from the origin to its correct location the axis of rotation can be considered to remain "embedded" in the part.
- *Rotation angles* are in degrees, so for a rotary joint with a 0-1 scaling you would need to use an angle scale of 360.

11.4.6 Assembling the model.

In order for parts to move together they need to be assembled with the `Collection()` command. It is important to assemble the parts and define their motions in the correct sequence. For example to create a moving head milling machine with a rotating spindle and an animated draw bar you would:

- Create the head main body.
- Create the spindle at the origin.
- Define the rotation.
- Move the head to the spindle or spindle to the head.
- Create the draw bar
- Define the motion of the draw bar
- Assemble the three parts into a head assembly
- Define the motion of the head assembly.

In this example the spindle rotation is indicated by rotation of a set of drive dogs:

```
#Drive dogs
dogs = Box(-6,-3,94,6,3,100)
dogs = Color([1,1,1,1],[dogs])
dogs = HalRotate([dogs],c,"spindle",360,0,0,1)
dogs = Translate([dogs],-1,49,0)

#Drawbar
draw = CylinderZ(120,3,125,3)
draw = Color([1,0,.5,1],[draw])
draw = Translate([draw],-1,49,0)
draw = HalTranslate([draw],c,"drawbar",0,0,1)

# head/spindle
head = AsciiSTL(filename="./head.stl")
head = Color([0.3,0.3,0.3,1],[head])
head = Translate([head],0,0,4)
head = Collection([head, tool, dogs, draw])
head = HalTranslate([head],c,"Z",0,0,0.1)

# base
```

```
base = AsciiSTL(filename="./base.stl")
base = Color([0.5,0.5,0.5,1],[base])
# mount head on it
base = Collection([head, base])
```

Finally a single collection of all the machine parts, floor and work (if any) needs to be created:

- For a *serial machine* each new part will be added to the collection of the previous part.
- For a *parallel machine* there may be several "base" parts.

Thus, for example, in scaragui.py link3 is added to link2, link2 to link1 and link1 to link0, so the final model is created by:

```
model = Collection([link0, floor, table])
```

Whereas a VMC model with separate parts moving on the base might have:

```
model = Collection([base, saddle, head, carousel])
```

11.4.7 Other functions

- `part = Color([colourspec], [part])`
Sets the display color of the part. Note that unlike the other functions the part definition comes second in this case.
The colourspec consists of the three RGB values and an opacity.
For example [1,0,0,0.5] for a 50% opacity red.
- `myhud = Hud()`
Creates a heads-up display in the Vismach GUI to display such items as axis positions.
- `part = Capture()`
I have no idea what this does ! But it seems to be important for tool tip visualization...
- `main(model, tooltip, work, size=10, hud=0, rotation_vectors=None, lat=0, lon=0)`
This is the command that makes it all happen, creates the display etc.
 - *model* should be a collection that contains all the machine parts.
 - *tooltip* and *work* need to be created by `Capture()` to visualize their motion in the back plot.
See scaragui.py for an example of how to connect the tool tip to a tool and the tool to the model.
 - Either *rotation_vectors* or *latitude/longitude* can be used to set the original viewpoint and it is advisable to do as the default initial viewpoint is rather unhelpfully from immediately overhead.
 - *size* sets the extent of the volume visualized in the initial view.
 - *hud* refers to a head-up display of axis positions.

11.4.8 Basic structure of a Vismach script.

```
#imports
from vismach import *
import hal
#create the HAL component and pins
comp = hal.component("compname")
comp.newpin("pin_name", hal.HAL_FLOAT, hal.HAL_IN)
...
#create the floor, tool and work
```

```
floor = Box(-50, -50, -3, 50, 50, 0)
work = Capture()
tooltip = Capture()
...
#Build and assemble the model
part1 = Collection([Box(-6,-3,94,6,3,100)])
part1 = Color([1,1,1,1],[part1])
part1 = HalRotate([part1],comp,"pin_name",360,0,0,1)
part1 = Translate([dogs],-1,49,0)
...
#create a top-level model
model = Collection([base, saddle, head, carousel])
#Start the visualization
main(model, tooltip, work, 100, lat=-75, lon=215)
```

11.5 Python Interface

This documentation describes the *linuxcnc* python module, which provides a python API for talking to LinuxCNC.

11.5.1 The linuxcnc Python module

User interfaces control LinuxCNC activity by sending NML messages to the LinuxCNC task controller, and monitor results by observing the LinuxCNC status structure, as well as the error reporting channel.

Programmatic access to NML is through a C++ API; however, the most important parts of the NML interface to LinuxCNC are also available to Python programs through the *linuxcnc* module.

Beyond the NML interface to the command, status and error channels, the *linuxcnc* module also contains:

- support for reading values from INI files

11.5.2 Usage Patterns for the LinuxCNC NML interface

The general pattern for *linuxcnc* usage is roughly like this:

- import the *linuxcnc* module
- establish connections to the command, status and error NML channels as needed
- poll the status channel, either periodically or as needed
- before sending a command, determine from status whether it is in fact OK to do so (for instance, there is no point in sending a *Run* command if task is in the ESTOP state, or the interpreter is not idle)
- send the command by using one of the *linuxcnc* command channel methods

To retrieve messages from the error channel, poll the error channel periodically, and process any messages retrieved.

- poll the status channel, either periodically or as needed
- print any error message and explore the exception code

linuxcnc also defines the error Python exception type to support error reporting.

11.5.3 Reading LinuxCNC status

Here is a Python fragment to explore the contents of the `linuxcnc.stat` object which contains some 80+ values (run while linuxcnc is running for typical values):

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
import sys
import linuxcnc
try:
    s = linuxcnc.stat() # create a connection to the status channel
    s.poll() # get current values
except linuxcnc.error, detail:
    print("error", detail)
    sys.exit(1)
for x in dir(s):
    if not x.startswith("_"):
        print(x, getattr(s,x))
```

Linuxcnc uses the default compiled-in path to the NML configuration file unless overridden, see [Reading INI file values](#) for an example.

11.5.3.1 linuxcnc.stat attributes

acceleration

(returns float) - default acceleration, reflects the INI entry [TRAJ]DEFAULT_ACCELERATION.

active_queue

(returns integer) - number of motions blending.

actual_position

(returns tuple of floats) - current trajectory position, (x y z a b c u v w) in machine units.

adaptive_feed_enabled

(returns boolean) - status of adaptive feedrate override (0/1).

ain

(returns tuple of floats) - current value of the analog input pins.

angular_units

(returns float) - machine angular units per deg, reflects [TRAJ]ANGULAR_UNITS INI value.

about

(returns tuple of floats) - current value of the analog output pins.

axes

(returns integer) - number of axes. Derived from [TRAJ]COORDINATES INI value.

axis

(returns tuple of dicts) - reflecting current axis values. See [The axis dictionary](#).

axis_mask

(returns integer) - mask of axis available as defined by [TRAJ]COORDINATES in the INI file. Returns the sum of the axes X=1, Y=2, Z=4, A=8, B=16, C=32, U=64, V=128, W=256.

block_delete

(returns boolean) - block delete current status.

call_level

(returns integer) - current subroutine depth. - 0 If not in a subroutine, Depth if not otherwise specified

command

(returns string) - currently executing command.

current_line

(returns integer) - currently executing line.

current_vel

(returns float) - current velocity in user units per second.

cycle_time

(returns float) - thread period

debug

(returns integer) - debug flag from the INI file.

delay_left

(returns float) - remaining time on dwell (G4) command, seconds.

din

(returns tuple of integers) - current value of the digital input pins.

distance_to_go

(returns float) - remaining distance of current move, as reported by trajectory planner.

dout

(returns tuple of integers) - current value of the digital output pins.

dtg

(returns tuple of floats) - remaining distance of current move for each axis, as reported by trajectory planner.

echo_serial_number

(returns integer) - The serial number of the last completed command sent by a UI to task. All commands carry a serial number. Once the command has been executed, its serial number is reflected in echo_serial_number.

enabled

(returns boolean) - trajectory planner enabled flag.

estop

(returns integer) - Returns either STATE_ESTOP or not.

exec_state

(returns integer) - task execution state. One of EXEC_ERROR, EXEC_DONE, EXEC_WAITING_FOR_MOTION, EXEC_WAITING_FOR_MOTION_QUEUE, EXEC_WAITING_FOR_IO, EXEC_WAITING_FOR_MOTION_A, EXEC_WAITING_FOR_DELAY, EXEC_WAITING_FOR_SYSTEM_CMD, EXEC_WAITING_FOR_SPINDLE.

feed_hold_enabled

(returns boolean) - enable flag for feed hold.

feed_override_enabled

(returns boolean) - enable flag for feed override.

feedrate

(returns float) - current feedrate override, 1.0 = 100%.

file

(returns string) - currently loaded G-code filename with path.

flood

(returns integer) - Flood status, either FLOOD_OFF or FLOOD_ON.

g5x_index

(returns integer) - currently active coordinate system, G54=1, G55=2 etc.

g5x_offset

(returns tuple of floats) - offset of the currently active coordinate system.

g92_offset

(returns tuple of floats) - pose of the current g92 offset.

gcodes

(returns tuple of integers) - Active G-codes for each modal group. G-code constants G_0, G_1, G_2, G_3, G_4, G_5, G_5_1, G_5_2, G_5_3, G_7, G_8, G_100, G_17, G_17_1, G_18, G_18_1, G_19, G_19_1, G_20, G_21, G_28, G_28_1, G_30, G_30_1, G_33, G_33_1, G_38_2, G_38_3, G_38_4, G_38_5, G_40, G_41, G_41_1, G_42, G_42_1, G_43, G_43_1, G_43_2, G_49, G_50, G_51, G_53, G_54, G_55, G_56, G_57, G_58, G_59, G_59_1, G_59_2, G_59_3, G_61, G_61_1, G_64, G_73, G_76, G_80, G_81, G_82, G_83, G_84, G_85, G_86, G_87, G_88, G_89, G_90, G_90_1, G_91, G_91_1, G_92, G_92_1, G_92_2, G_92_3, G_93, G_94, G_95, G_96, G_97, G_98, G_99

homed

(returns tuple of integers) - currently homed joints, 0 = not homed, 1 = homed.

id

(returns integer) - currently executing motion id.

ini_filename

(returns string) - path to the INI file passed to linuxcnc.

inpos

(returns boolean) - machine-in-position flag.

input_timeout

(returns boolean) - flag for M66 timer in progress.

interp_state

(returns integer) - current state of RS274NGC interpreter. One of INTERP_IDLE, INTERP_READING, INTERP_PAUSED, INTERP_WAITING.

interpreter_errcode

(returns integer) - current RS274NGC interpreter return code. One of INTERP_OK, INTERP_EXIT, INTERP_EXECUTE_FINISH, INTERP_ENDFILE, INTERP_FILE_NOT_OPEN, INTERP_ERROR. see src/emc/nml_intf/interp_return.hh

joint

(returns tuple of dicts) - reflecting current joint values. See [The joint dictionary](#).

joint_actual_position

(returns tuple of floats) - actual joint positions.

joint_position

(returns tuple of floats) - Desired joint positions.

joints

(returns integer) - number of joints. Reflects [KINS]JOINTS INI value.

kinematics_type

(returns integer) - The type of kinematics. One of:

- KINEMATICS_IDENTITY
- KINEMATICS_FORWARD_ONLY
- KINEMATICS_INVERSE_ONLY
- KINEMATICS_BOTH

limit

(returns tuple of integers) - axis limit masks. minHardLimit=1, maxHardLimit=2, minSoftLimit=4, maxSoftLimit=8.

linear_units

(returns float) - machine linear units per mm, reflects [TRAJ]LINEAR_UNITS INI value.

lube

(returns integer) - lube on flag.

lube_level

(returns integer) - reflects iocontrol.0.lube_level.

max_acceleration

(returns float) - maximum acceleration. Reflects [TRAJ]MAX_ACCELERATION.

max_velocity

(returns float) - maximum velocity. Reflects the current maximum velocity. If not modified by halui.max-velocity or similar it should reflect [TRAJ]MAX_VELOCITY.

mcodes

(returns tuple of 10 integers) - currently active M-codes.

mist

(returns integer) - Mist status, either MIST_OFF or MIST_ON

motion_line

(returns integer) - source line number motion is currently executing. Relation to id unclear.

motion_mode

(returns integer) - This is the mode of the Motion controller. One of TRAJ_MODE_COORD, TRAJ_MODE_FREE, TRAJ_MODE_TELEOP.

motion_type

(returns integer) - The type of the currently executing motion. One of:

- MOTION_TYPE_TRAVERSE
- MOTION_TYPE_FEED
- MOTION_TYPE_ARC
- MOTION_TYPE_TOOLCHANGE
- MOTION_TYPE_PROBING
- MOTION_TYPE_INDEXROTARY
- Or 0 if no motion is currently taking place.

optional_stop

(returns integer) - option stop flag.

paused

(returns boolean) - motion paused flag.

pocket_prepped

(returns integer) - A Tx command completed, and this pocket is prepared. -1 if no prepared pocket.

poll()

-(built-in function) method to update current status attributes.

position

(returns tuple of floats) - trajectory position.

probe_tripped

(returns boolean) - flag, True if probe has tripped (latch)

probe_val

(returns integer) - reflects value of the motion.probe-input pin.

probed_position

(returns tuple of floats) - position where probe tripped.

probing

(returns boolean) - flag, True if a probe operation is in progress.

program_units

(returns integer) - one of CANON_UNITS_INCHES=1, CANON_UNITS_MM=2, CANON_UNITS_CM=3

queue

(returns integer) - current size of the trajectory planner queue.

queue_full

(returns boolean) - the trajectory planner queue is full.

rapidrate

(returns float) - rapid override scale.

read_line

(returns integer) - line the RS274NGC interpreter is currently reading.

rotation_xy

(returns float) - current XY rotation angle around Z axis.

settings

(returns tuple of floats) - current interpreter settings. settings[0] = sequence number, settings[1] = feed rate, settings[2] = speed, settings[3] = G64 P blend tolerance, settings[4] = G64 Q naive CAM tolerance.

spindle

' (returns tuple of dicts) ' - returns the current spindle status, see [The spindle dictionary](#)

spindles

(returns integer) - number of spindles. Reflects [TRAJ]SPINDLES INI value.

state

(returns integer) - current command execution status. One of RCS_DONE, RCS_EXEC, RCS_ERROR.

task_mode

(returns integer) - current task mode. one of MODE_MDI, MODE_AUTO, MODE_MANUAL.

task_paused

(returns integer) - task paused flag.

task_state

(returns integer) - current task state. one of STATE_ESTOP, STATE_ESTOP_RESET, STATE_ON, STATE_OFF.

tool_in_spindle

(returns integer) - current tool number.

tool_from_pocket

(returns integer) - pocket number for the currently loaded tool (0 if no tool loaded).

tool_offset

(returns tuple of floats) - offset values of the current tool.

tool_table

(returns tuple of tool_results) - list of tool entries. Each entry is a sequence of the following fields: id, xoffset, yoffset, zoffset, aoffset, boffset, coffset, uoffset, voffset, woffset, diameter, frontangle, backangle, orientation. The id and orientation are integers and the rest are floats.

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
import linuxcnc
s = linuxcnc.stat()
s.poll()
# to find the loaded tool information it is in tool table index 0
if s.tool_table[0].id != 0: # a tool is loaded
    print(s.tool_table[0].zoffset)
else:
    print("No tool loaded.")
```

velocity

(returns float) - This property is defined, but it does not have a useful interpretation.

11.5.3.2 The axis dictionary

The axis configuration and status values are available through a list of per-axis dictionaries. Here's an example how to access an attribute of a particular axis: Note that many properties that were formerly in the axis dictionary are now in the joint dictionary, because on nontrivial kinematics machines these items (such as backlash) are not the properties of an axis.

max_position_limit

(returns float) - maximum limit (soft limit) for axis motion, in machine units.configuration parameter, reflects [JOINT_n]MAX_LIMIT.

min_position_limit

(returns float) - minimum limit (soft limit) for axis motion, in machine units.configuration parameter, reflects [JOINT_n]MIN_LIMIT.

velocity

(returns float) - current velocity.

11.5.3.3 The joint dictionary

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
import linuxcnc
s = linuxcnc.stat()
s.poll()
print("Joint 1 homed: ", s.joint[1]["homed"])
```

For each joint, the following dictionary keys are available:

backlash

(returns float) - Backlash in machine units. configuration parameter, reflects [JOINT_n]BACKLASH.

enabled

(returns integer) - non-zero means enabled.

fault

(returns integer) - non-zero means axis amp fault.

error_current

(returns float) - current following error.

ferror_highmark

(returns float) - magnitude of max following error.

homed

(returns integer) - non-zero means has been homed.

homing

(returns integer) - non-zero means homing in progress.

inpos

(returns integer) - non-zero means in position.

input

(returns float) - current input position.

jointType

(returns integer) - type of axis configuration parameter, reflects [JOINT_n]TYPE. LINEAR=1, ANGULAR=2. See [Joint INI configuration](#) for details.

max_ferror

(returns float) - maximum following error. configuration parameter, reflects [JOINT_n]FERROR.

max_hard_limit

(returns integer) - non-zero means max hard limit exceeded.

max_position_limit

(returns float) - maximum limit (soft limit) for joint motion, in machine units. configuration parameter, reflects [JOINT_n]MAX_LIMIT.

max_soft_limit

non-zero means max_position_limit was exceeded, int

min_ferror

(returns float) - configuration parameter, reflects [JOINT_n]MIN_FERROR.

min_hard_limit

(returns integer) - non-zero means min hard limit exceeded.

min_position_limit

(returns float) - minimum limit (soft limit) for joint motion, in machine units. configuration parameter, reflects [JOINT_n]MIN_LIMIT.

min_soft_limit

(returns integer) - non-zero means min_position_limit was exceeded.

output

(returns float) - commanded output position.

override_limits

(returns integer) - non-zero means limits are overridden.

units

(returns float) - joint units per mm, or per degree for angular joints.

(joint units are the same as machine units, unless set otherwise by the configuration parameter [JOINT_n]UNITS)

velocity

(returns float) - current velocity.

11.5.4 The spindle dictionary

brake

(returns integer) - value of the spindle brake flag.

direction

(returns integer) - rotational direction of the spindle. forward=1, reverse=-1.

enabled

(returns integer) - value of the spindle enabled flag.

homed

(not currently implemented)

increasing

(returns integer) - unclear.

orient_fault

(returns integer)

orient_state

(returns integer)

override

(returns float) - spindle speed override scale.

override_enabled

(returns boolean) - value of the spindle override enabled flag.

speed

(returns float) - spindle speed value, rpm, > 0: clockwise, < 0: counterclockwise.

11.5.5 Preparing to send commands

Some commands can always be sent, regardless of mode and state; for instance, the `linuxcnc.command.abort` method can always be called.

Other commands may be sent only in appropriate state, and those tests can be a bit tricky. For instance, an MDI command can be sent only if:

- ESTOP has not been triggered, and
- the machine is turned on and
- the axes are homed and
- the interpreter is not running and
- the mode is set to MDI mode

so an appropriate test before sending an MDI command through `linuxcnc.command.mdi()` could be:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
import linuxcnc
s = linuxcnc.stat()
c = linuxcnc.command()

def ok_for_mdi():
    s.poll()
```

```

    return not s.estop and s.enabled and (s.homed.count(1) == s.joints) and (s.interp_state <-
        == linuxcnc.INTERP_IDLE)

if ok_for_mdi():
    c.mode(linuxcnc.MODE_MDI)
    c.wait_complete() # wait until mode switch executed
    c.mdi("G0 X10 Y20 Z30")

```

11.5.6 Sending commands through linuxcnc.command

Before sending a command, initialize a command channel like so:

```

#!/usr/bin/env python3
# -*- coding: utf-8 -*-
import linuxcnc
c = linuxcnc.command()

# Usage examples for some of the commands listed below:
c.abort()

c.auto(linuxcnc.AUTO_RUN, program_start_line)
c.auto(linuxcnc.AUTO_STEP)
c.auto(linuxcnc.AUTO_PAUSE)
c.auto(linuxcnc.AUTO_RESUME)

c.brake(linuxcnc.BRAKE_ENGAGE)
c.brake(linuxcnc.BRAKE_RELEASE)

c.flood(linuxcnc.FLOOD_ON)
c.flood(linuxcnc.FLOOD_OFF)

c.home(2)

c.jog(linuxcnc.JOG_STOP,          jjogmode, joint_num_or_axis_index)
c.jog(linuxcnc.JOG_CONTINUOUS,   jjogmode, joint_num_or_axis_index, velocity)
c.jog(linuxcnc.JOG_INCREMENT,    jjogmode, joint_num_or_axis_index, velocity, increment)

c.load_tool_table()

c.maxvel(200.0)

c.mdi("G0 X10 Y20 Z30")

c.mist(linuxcnc.MIST_ON)
c.mist(linuxcnc.MIST_OFF)

c.mode(linuxcnc.MODE_MDI)
c.mode(linuxcnc.MODE_AUTO)
c.mode(linuxcnc.MODE_MANUAL)

c.override_limits()

c.program_open("foo.ngc")
c.reset_interpreter()

c.tool_offset(toolno, z_offset, x_offset, diameter, frontangle, backangle, orientation)

```

11.5.6.1 linuxcnc.command attributes**serial**

the current command serial number

11.5.6.2 linuxcnc.command methods:**abort()**

send EMC_TASK_ABORT message.

auto(int[, int])

run, step, pause or resume a program.

brake(int)

engage or release spindle brake.

debug(int)

set debug level via EMC_SET_DEBUG message.

display_msg(string)

sends a operator display message to the screen. (max 254 characters)

error_msg(string)

sends a operator error message to the screen. (max 254 characters)

feedrate(float)

set the feedrate.

flood(int)

turn on/off flooding.

Syntax

```
flood(command)
flood(linuxcnc.FLOOD_ON)
flood(linuxcnc.FLOOD_OFF)
```

Constants

```
FLOOD_ON
FLOOD_OFF
```

home(int)

home a given joint.

jog(command-constant, bool, int[, float[, float]])**Syntax**

```
jog(command, jjogmode, joint_num_or_axis_index, velocity[, distance])
jog(linuxcnc.JOG_STOP, jjogmode, joint_num_or_axis_index)
jog(linuxcnc.JOG_CONTINUOUS, jjogmode, joint_num_or_axis_index, velocity)
jog(linuxcnc.JOG_INCREMENT, jjogmode, joint_num_or_axis_index, velocity, distance)
```

Command Constants

```
linuxcnc.JOG_STOP
linuxcnc.JOG_CONTINUOUS
linuxcnc.JOG_INCREMENT
```

jjogmode**True**

request individual joint jog (requires teleop_enable(0))

False

request axis Cartesian coordinate jog (requires teleop_enable(1))

joint_num_or_axis_index**For joint jog (jjogmode=1)**

joint_number

For axis Cartesian coordinate jog (jjogmode=0)

zero-based index of the axis coordinate with respect to the known coordinate letters
XYZABCUVW (x=>0,y=>1,z=>2,a=>3,b=>4,c=>5,u=>6,v=>7,w=>8)

load_tool_table()

reload the tool table.

maxvel(float)

set maximum velocity

mdi(string)

send an MDI command. Maximum 254 chars.

mist(int)

turn on/off mist.

Syntax

mist(command)

mist(linuxcnc.MIST_ON)

mist(linuxcnc.MIST_OFF)

Constants

MIST_ON

MIST_OFF

mode(int)

set mode (MODE_MDI, MODE_MANUAL, MODE_AUTO).

override_limits()

set the override axis limits flag.

program_open(string)

open an NGC file.

rapidrate()

set rapid override factor

reset_interpreter()

reset the RS274NGC interpreter

set_adaptive_feed(int)

set adaptive feed flag

set_analog_output(int, float)

set analog output pin to value

set_block_delete(int)

set block delete flag

set_digital_output(int, int)

set digital output pin to value

set_feed_hold(int)

set feed hold on/off

set_feed_override(int)

set feed override on/off

set_max_limit(int, float)

set max position limit for a given axis

set_min_limit()

set min position limit for a given axis

set_optional_stop(int)

set optional stop on/off

set_spindle_override(int [, int])

set spindle override enabled. Defaults to spindle 0.

spindle(direction: int, speed: float=0, spindle: int=0, wait_for_speed: int=0)

- Direction: [SPINDLE_FORWARD, SPINDLE_REVERSE, SPINDLE_OFF, SPINDLE_INCREASE, SPINDLE_DECREASE, or SPINDLE_CONSTANT]
- Speed: Speed in RPM, defaults to 0.
- Spindle: Spindle number to command defaults to 0.
- Wait_for_speed: if 1 motion will wait for speed before continuing, defaults to not.

**Warning**

MDI commands will ignore this. "S1000" after this will turn the spindle off.

text_msg(string)

sends a operator text message to the screen. (max 254 characters)

```
#!/usr/bin/env python3
import linuxcnc
c = linuxcnc.command()

# Increase speed of spindle 0 by 100rpm. Spindle must be on first.
c.spindle(linuxcnc.INCREASE)

# Increase speed of spindle 2 by 100rpm. Spindle must be on first.
c.spindle(linuxcnc.SPINDLE_INCREASE, 2)

# Set speed of spindle 0 to 1024 rpm.
c.spindle.(linuxcnc.SPINDLE_FORWARD, 1024)

# Set speed of spindle 1 to -666 rpm.
c.spindle.(linuxcnc.SPINDLE_REVERSE, 666, 1)

# Stop spindle 0.
c.spindle.(linuxcnc.SPINDLE_OFF)

# Stop spindle 0 explicitly.
c.spindle.(linuxcnc.SPINDLE_OFF, 0)
```

spindleoverride(float [, int])

Set spindle override factor. Defaults to spindle 0.

state(int)

Set the machine state. Machine state should be STATE_ESTOP, STATE_ESTOP_RESET, STATE_ON, or STATE_OFF.

task_plan_sync()

On completion of this call, the var file on disk is updated with live values from the interpreter.

teleop_enable(int)

Enable/disable teleop mode (disable for joint jogging).

tool_offset(int, float, float, float, float, float, int)

Set the tool offset. See usage example above.

traj_mode(int)

Set trajectory mode. Mode is one of MODE_FREE, MODE_COORD, or MODE_TELEOP.

unhome(int)

Unhome a given joint.

wait_complete([float])

Wait for completion of the last command sent. If timeout in seconds not specified, default is 5 seconds. Return -1 if timed out, return RCS_DONE or RCS_ERROR according to command execution status.

11.5.7 Reading the error channel

To handle error messages, connect to the error channel and periodically poll() it.

Note that the NML channel for error messages has a queue (other than the command and status channels), which means that the first consumer of an error message deletes that message from the queue; whether your another error message consumer (e.g. Axis) will *see* the message is dependent on timing. It is recommended to have just one error channel reader task in a setup.

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
import linuxcnc
e = linuxcnc.error_channel()

error = e.poll()

if error:
    kind, text = error
    if kind in (linuxcnc.NML_ERROR, linuxcnc.OPERATOR_ERROR):
        typus = "error"
    else:
        typus = "info"
    print(typus, text)
```

11.5.8 Reading INI file values

Here's an example for reading values from an INI file through the linuxcnc.ini object:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
# run as:
# python3 ini-example.py ~/emc2-dev/configs/sim/axis/axis_mm.ini

import sys
import linuxcnc
```

```

infile = linuxcnc.ini(sys.argv[1])

# infile.find() returns None if the key wasn't found - the
# following idiom is useful for setting a default value:

machine_name = infile.find("EMC", "MACHINE") or "unknown"
print("machine name: ", machine_name)

# infile.findall() returns a list of matches, or an empty list
# if the key wasn't found:

extensions = infile.findall("FILTER", "PROGRAM_EXTENSION")
print("extensions: ", extensions)

# override default NML file by INI parameter if given
nmlfile = infile.find("EMC", "NML_FILE")
if nmlfile:
    linuxcnc.nmlfile = os.path.join(os.path.dirname(sys.argv[1]), nmlfile)

```

Or for the same INI file as LinuxCNC:

```

#!/usr/bin/env python3
# -*- coding: utf-8 -*-
# run as:
# python3 ini-example2.py

import linuxcnc

stat = linuxcnc.stat()
stat.poll()

infile = linuxcnc.ini(stat.ini_filename)

# See example above for usage of 'infile' object

```

11.5.9 The linuxcnc.positionlogger type

Some usage hints can be gleaned from `src/emc/usr_intf/gremlin/gremlin.py`.

11.5.9.1 members

npts
number of points.

11.5.9.2 methods

start(float)
start the position logger and run every ARG seconds

clear()
clear the position logger

stop()
stop the position logger

call()

Plot the backplot now.

last([int])

Return the most recent point on the plot or None

11.6 GStat

11.6.1 Intro

GStat is a python class used to send messages from LinuxCNC to other python programs. It uses GObject to deliver messages, making it easy to listen for specific information. This is referred to as event-driven programming, which is more efficient than every program polling LinuxCNC at the same time. GladeVCP, Gscreen, Gmoccapy and QtVCP use GStat extensively. GStat is in the `hal_glib` module.

11.6.2 Overview

First a program imports the `hal_glib` module and instantiates GStat. Then it *connects* to the messages it wishes to monitor. GStat checks LinuxCNC's status every 100ms and if there are differences from the last check, it will send a callback message to all the connected programs with the current status. When GStat calls the registered function, it sends the GStat object plus any return codes from the message. Typical code signatures:

```
GSTAT.connect('MESSGAE-TO-LISTEN-FOR', FUNCTION_TO_CALL)
```

```
def FUNCTION_TO_CALL(gstat_object, return_codes):
```

Often LAMBDA is used to strip the GSTAT object and manipulate the return codes:

```
GSTAT.connect('MESSGAE-TO-LISTEN-FOR', lambda o, return: FUNCTION_TO_CALL(not return))
```

```
def FUNCTION_TO_CALL(return_codes):
```

11.6.3 Sample GStat Code

There are some basic patterns for using GStat, depending on what library you are using them in. If using GStat with GladeVCP, Gscreen, or Qtvcp, the gobject library is not needed as those toolkits already set up GObject.

11.6.3.1 Sample HAL component code pattern

This program creates two HAL pins that output the status of G20/G21.

```
#!/usr/bin/env python3

import gi
gi.require_version('Gtk', '3.0')
from gi.repository import GObject
from gi.repository import GLib
import hal
from hal_glib import GStat
GSTAT = GStat()

# callback to change HAL pin state
def mode_changed(obj, data):
    h['g20'] = not data
    h['g21'] = data

# Make a component and pins
h = hal.component("metric_status")
h.newpin("g20", hal.HAL_BIT, hal.HAL_OUT)
h.newpin("g21", hal.HAL_BIT, hal.HAL_OUT)
h.ready()

# connect a GSTAT message to a callback function
GSTAT.connect("metric-mode-changed", mode_changed)

# force GSTAT to initialize states
GSTAT.forced_update()

# loop till exit
try:
    GLib.MainLoop().run()
except KeyboardInterrupt:
    raise SystemExit
```

This would be loaded with *loadusr python PATH-TO-FILE/FILENAME.py*
or if you need to wait for the pins to be made before continuing:
loadusr python -Wn metric_status PATH-TO-FILE/FILENAME.py
The pins would be: *metric_status.g20* and *metric_status.g21*.

11.6.3.2 GladeVCP python extension code pattern

This file assumes there are three GTK labels named:

- *state_label*
- *e_state_label*
- *interp_state_label*

```
#!/usr/bin/env python3

from hal_glib import GStat
GSTAT = GStat()

class HandlerClass:

    def __init__(self, halcomp, builder, useropts):
        self.builder = builder

        GSTAT.connect("state-estop", lambda w: self.update_estate_label('ESTOP'))
```

```

GSTAT.connect("state-estop-reset",lambda w: self.update_estate_label('RESET'))

GSTAT.connect("state-on",lambda w: self.update_state_label('MACHIBE ON'))
GSTAT.connect("state-off",lambda w: self.update_state_label('MACHINE OFF'))

GSTAT.connect("interp-paused",lambda w: self.update_interp_label('Paused'))
GSTAT.connect("interp-run",lambda w: self.update_interp_label('Run'))
GSTAT.connect("interp-idle",lambda w: self.update_interp_label('Idle'))

def update_state_label(self,text):
    self.builder.get_object('state_label').set_label("State: %s" % (text))

def update_estate_label(self,text):
    self.builder.get_object('e_state_label').set_label("E State: %s" % (text))

def update_interp_label(self,text):
    self.builder.get_object('interp_state_label').set_label("Interpreter State: %s" % ( ←
        text))

def get_handlers(halcomp,builder,useropts):
    return [HandlerClass(halcomp,builder,useropts)]

```

11.6.3.3 QTVCP python extension code pattern

Qtvcp extends GStat, so must be loaded differently but all the messages are available in Qtvcp. This handler file assumes there are three QLabels named:

- *state_label*
- *e_state_label*
- *interp_state_label*

```

#!/usr/bin/env python3

from qtvcp.core import Status
GSTAT = Status()

class HandlerClass:

    def __init__(self, halcomp,widgets,paths):
        self.w = widgets

        GSTAT.connect("state-estop",lambda w: self.update_estate_label('ESTOP'))
        GSTAT.connect("state-estop-reset",lambda w: self.update_estate_label('RESET'))

        GSTAT.connect("state-on",lambda w: self.update_state_label('MACHIBE ON'))
        GSTAT.connect("state-off",lambda w: self.update_state_label('MACHINE OFF'))

        GSTAT.connect("interp-paused",lambda w: self.update_interp_label('Paused'))
        GSTAT.connect("interp-run",lambda w: self.update_interp_label('Run'))
        GSTAT.connect("interp-idle",lambda w: self.update_interp_label('Idle'))

    def update_state_label(self,text):
        self.w.state_label.setText("State: %s" % (text))

    def update_estate_label(self,text):
        self.w.e_state_label.setText("E State: %s" % (text))

```

```
def update_interp_label(self, text):
    self.winterp_state_label.setText("Interpreter State: %s" % (text))

def get_handlers(halcomp, builder, useropts):
    return [HandlerClass(halcomp, widgets, paths)]
```

11.6.4 Messages

periodic

(returns nothing) - sent every 100 ms.

state-estop

(returns nothing) - Sent when linuxcnc is goes into estop.

state-estop-reset

(returns nothing) - Sent when linuxcnc comes out of estop.

state-on

(returns nothing) - Sent when linuxcnc is in machine on state.

state-off

(returns nothing) - Sent when linuxcnc is in machine off state.

homed

(returns string) - Sent as each joint is homed.

all-homed

(returns nothing) - Sent when all defined joints are homed.

not-all-homed

(returns string) - Sends a list of joints not currently homed.

override_limits_changed

(returns string) - Sent if linuxcnc has been directed to override it's limits.

hard-limits-tripped

(returns bool, Python List) - Sent when any hard limit is tripped. bool indicates if any limit is tripped, the list shows all available joint's current limit values.

mode-manual

(returns nothing) - Sent when linuxcnc switches to manual mode.

mode-mdi

(returns nothing) - Sent when linuxcnc switches to mdi mode.

mode-auto

(returns nothing) - Sent when linuxcnc switches to auto mode.

command-running

(returns nothing) - Sent when running a program or MDI

command-stopped

(returns nothing) - Sent when a program or MDI stopped

command-error

(returns nothing) - Sent when there is a command error

interp-run

(returns nothing) - Sent when linuxcnc's interpreter is running an MDI or program.

interp-idle

(returns nothing) - Sent when linuxcnc's interpreter idle.

interp-paused

(returns nothing) - Sent when linuxcnc's interpreter is paused.

interp-reading

(returns nothing) - Sent when linuxcnc's interpreter is reading.

interp-waiting

(returns nothing) - Sent when linuxcnc's interpreter is waiting.

jograte-changed

(returns float) - Sent when jog rate has changed.

Linuxcnc does not have an internal jog rate.

This is GStat's internal jog rate.

It is expected to be in the machine's native units regardless of the current unit mode .

jograte-angular-changed

(returns float) - Sent when the angular jog rate has changed.

Linuxcnc does not have an internal angular jog rate.

This is GStat's internal jog rate.

It is expected to be in the machine's native units regardless of the current unit mode .

jogincrement-changed

(returns float, text) - Sent when jog increment has changed.

Linuxcnc does not have an internal jog increment.

This is GStat's internal jog increment.

It is expected to be in the machine's native units regardless of the current unit mode .

jogincrement-angular-changed

(returns float, text) - Sent when angular jog increment has changed.

Linuxcnc does not have an internal angular jog increment.

This is GStat's internal angular jog increment.

It is expected to be in the machine's native units regardless of the current unit mode .

program-pause-changed

(returns bool) - Sent when program is paused/unpaused.

optional-stop-changed

(returns bool) - Sent when optional stop is set/unset

block-delete-changed

(returns bool) - sent when block delete is set/unset.

file-loaded

(returns string) - Sent when linuxcnc has loaded a file

reload-display

(returns nothing) - Sent when there is a request to reload the display

line-changed

(returns integer) - Sent when linuxcnc has read a new line.

Linuxcnc does not update this for every type of line.

tool-in-spindle-changed

(returns integer) - Sent when the tool has changed.

tool-info-changed

(returns python object) - Sent when current tool info changes.

current-tool-offset

(returns python object) - Sent when the current tool offsets change.

motion-mode-changed

(returns integer) - Sent when motion's mode has changed

spindle-control-changed

(returns integer, bool, integer, bool) - (spindle num, spindle on state, requested spindle direction & rate, at-speed state)

Sent when spindle direction or running status changes or at-speed changes.

current-feed-rate

(returns float) - Sent when the current feed rate changes.

current-x-rel-position

(returns float) - Sent every 100ms.

current-position

(returns pyobject, pyobject, pyobject, pyobject) - Sent every 100ms.

returns tuples of position, relative position, distance-to-go and the joint actual position. Before homing, on multi-joint axes, only joint position is valid.

current-z-rotation

(returns float) - Sent as the current rotated angle around the Z axis changes

requested-spindle-speed-changed

(returns float) - Sent when the current requested RPM changes

actual-spindle-speed-changed

(returns float) - Sent when the actual RPM changes based on the HAL pin *spindle.0.speed-in*.

spindle-override-changed

(returns float) - Sent when the spindle override value changes
in percent

feed-override-changed

(returns float) - Sent when the feed override value changes
in percent

rapid-override-changed

(returns float) - Sent when the rapid override value changes
in percent 0-100

max-velocity-override-changed

(returns float) - Sent when the maximum velocity override value changes
in units per minute

feed-hold-enabled-changed

(returns bool) - Sent when feed hold status changes

itime-mode

(returns bool) - Sent when G93 status changes
(inverse time mode)

fpm-mode

(returns bool) - Sent when G94 status changes
(feed per minute mode)

fpr-mode

(returns bool) - Sent when G95 status changes
(feed per revolution mode)

css-mode

(returns bool) - Sent when G96 status changes
(constant surface feed mode)

rpm-mode

(returns bool) - Sent when G97 status changes
(constant RPM mode)

radius-mode

(returns bool) - Sent when G8 status changes
display X in radius mode

diameter-mode

(returns bool) - Sent when G7 status changes
display X in Diameter mode

flood-changed

(returns bool) - Sent when flood coolant state changes.

mist-changed

(returns bool) - Sent when mist coolant state changes.

m-code-changed

(returns string) - Sent when active M-codes change

g-code-changed

(returns string) - Sent when active G-code change

metric-mode-changed

(returns bool) - Sent when G21 status changes

user-system-changed

(returns string) - Sent when the reference coordinate system (G5x) changes

mdi-line-selected

(returns string, string) - intended to be sent when an MDI line is selected by user.
This depends on the widget/libraries used.

gcode-line-selected

(returns integer) - intended to be sent when a G-code line is selected by user.
This depends on the widget/libraries used.

graphics-line-selected

(returns integer) - intended to be sent when graphics line is selected by user.
This depends on the widget/libraries used.

graphics-loading-progress

(returns integer) - intended to return percentage done of loading a program or running a program.
This depends on the widget/libraries used.

graphics-gcode-error

(returns string) - intended to be sent when a G-code error is found when loading.
This depends on the widget/libraries used.

graphics-gcode-properties

(returns string) - intended to be sent when G-code is loaded.
This depends on the widget/libraries used.

graphics-view-changed

(returns string, python Dict or None) - intended to be sent when graphics view is changed.
This depends on the widget/libraries used.

mdi-history-changed

(returns None) - intended to be sent when an MDI history needs to be reloaded.
This depends on the widget/libraries used.

machine-log-changed

(returns None) - intended to be sent when machine log has changed.
This depends on the widget/libraries used.

update-machine-log

(returns string, string) - intended to be sent when updating the machine.
This depends on the widget/libraries used.

move-text-lineup

(returns None) - intended to be sent when moving the cursor one line up in G-code display.
This depends on the widget/libraries used.

move-text-linedown

(returns None) - intended to be sent when moving the cursor one line down in G-code display.
This depends on the widget/libraries used.

dialog-request

(returns python dict) - intended to be sent when requesting a gui dialog.
It uses a python dict for communication.
The dict must include the following keyname pair:

- NAME: *requested dialog name*
The dict usually has several keyname pairs - it depends on the dialog.
dialogs return information using a general message
This depends on the widget/libraries used.

focus-overlay-changed

(returns bool, string, python object) - intended to be sent when requesting an overlay to be put over the display.
This depends on the widget/libraries used.

play-sound

(returns string) - intended to be sent when requesting a specific sound file to be played.
This depends on the widget/libraries used.

virtual-keyboard

(returns string) - intended to be sent when requesting a on screen keyboard.
This depends on the widget/libraries used.

dro-reference-change-request

(returns integer) - intended to be sent when requesting a DRO widget to change it's reference.
0 = machine, 1 = relative, 3 = distance-to-go
This depends on the widget/libraries used.

show-preferences

(returns None) - intended to be sent when requesting the screen preferences to be displayed.
This depends on the widget/libraries used.

shutdown

(returns None) - intended to be sent when requesting linuxcnc to shutdown.
This depends on the widget/libraries used.

error

(returns integer, string) - intended to be sent when an error has been reported .
integer represents the kind of error. ERROR, TEXT or DISPLAY
string is the actual error message.
This depends on the widget/libraries used.

general

(returns python dict) - intended to be sent when message must be sent that is not covered by a more specific message.
General message should be used a sparsely as reasonable because all object connected to it will have to parse it.
It uses a python dict for communication.
The dict should include and be checked for a unique id keyname pair:

- ID: *UNIQUE_ID_CODE*
The dict usually has more keyname pair - it depends on implementation.

forced-update

(returns None) - intended to be sent when one wishes to initialize or arbitrarily update an object.
This depends on the widget/libraries used.

progress

(returns integer, python object) intended to be sent to indicate the progress of a filter program.
This depends on the widget/libraries used.

following-error

(returns python list) returns a list of all joints current following error

11.6.5 Functions

These are convenience functions that are commonly used in programming.

set_jograte

(float) - Linuxcnc has no internal concept of jog rate -each GUI has it's own.
This is not always convenient.

This function allows one to set a jog rate for all objects connected to the signal *jograte-changed*.

It defaults to 15

GSTAT.set_jog_rate(10) would set the jog rate to 10 machine-units-per-minute and emit the jograte-changed signal.

get_jograte()

(*Nothing*) - x = GSTAT.get_jograte() would return GSTAT's current internal jograte (float).

set_jograte_angular

(*float*) -

get_jograte_angular

(*None*) -

set_jog_increment_angular

(*float, string*) -

get_jog_increment_angular

(*None*) -

set_jog_increments

(*float, string*) -

get_jog_increments

(*None*) -

is_all_homed

(*nothing*) - This will return the current state of all_homed (BOOL).

machine_is_on

(*nothing*) - This will return the current state of machine (BOOL).

estop_is_clear

(*nothing*) - This will return the state of Estop (BOOL)

set_tool_touchoff

(*tool,axis,value*) - This command will record the current mode, switch to MDI mode, invoke the MDI command: G10 L10 P[TOOL] [AXIS] [VALUE] wait for it to complete invoke G43 wait for it to complete switch back to the original mode.

set_axis_origin

(*axis,value*) - This command will record the current mode, switch to MDI mode, invoke the MDI command: G10 L20 P0 [AXIS] [VALUE] wait for it to complete switch back to the original mode. emit a *reload-display* signal.

do_jog

(*axis_number,direction, distance*) - This will jog an axis continuously or at a set distance. You must be in the proper mode to jog.

check_for_modes

(*mode*) - This function checks for required linuxcnc mode. It returns a python tuple (state, mode) mode will be set the mode the system is in

state will set to:
false if mode is 0
false if machine is busy
true if linuxcnc is in the requested mode
None if possible to change, but not in requested mode

get_current_mode

(nothing) - returns integer: the current linuxcnc mode.

set_selected_joint

(integer) - records the selected joint number internally.
requests the joint to be selected by emitting the
joint-selection-changed message.

get_selected_joint

(None) - returns integer representing the internal selected joint number.

set_selected_axis

(string) - records the selected axis letter internally.
requests the axis to be selected by emitting the
axis-selection-changed message.

get_selected_axis

(None) - returns string representing the internal selected axis letter.

is_man_mode

(None) -

is_mdi_mode

(None) -

is_auto_mode

(None) -

is_on_and_idle

(None) -

is_auto_running

(None) -

is_auto_paused

(None) -

is_file_loaded

(None) -

is_metric_mode

(None) -

is_spindle_on

(None) -

shutdown

(None) -

11.6.6 Known Issues

Some status points are reported wrongly during a running program.
This is because the interpreter runs ahead of the current position of a running program.
This will hopefully be resolved with the merge of state-tags branch.

Chapter 12

Drivers

12.1 Parallel Port Driver

The `hal_parport` component is a driver for the traditional PC parallel port. The port has a total of 17 physical pins. The original parallel port divided those pins into three groups: data, control, and status. The data group consists of 8 output pins, the control group consists of 4 pins, and the status group consists of 5 input pins.

In the early 1990's, the bidirectional parallel port was introduced, which allows the data group to be used for output or input. The HAL driver supports the bidirectional port, and allows the user to set the data group as either input or output. If configured as *out*, a port provides a total of 12 outputs and 5 inputs. If configured as *in*, it provides 4 outputs and 13 inputs.

In some parallel ports, the control group pins are open collectors, which may also be driven low by an external gate. On a board with open collector control pins. If configured as *x*, it provides 8 outputs, and 9 inputs.

In some parallel ports, the control group has push-pull drivers and cannot be used as an input.

HAL and Open Collectors

HAL cannot automatically determine if the *x* mode bidirectional pins are actually open collectors (OC). If they are not, they cannot be used as inputs, and attempting to drive them LOW from an external source can damage the hardware.

To determine whether your port has *open collector* pins, load `hal_parport` in *x* mode. With no device attached, HAL should read the pin as TRUE. Next, insert a 470 ohm resistor from one of the control pins to GND. If the resulting voltage on the control pin is close to 0V, and HAL now reads the pin as FALSE, then you have an OC port. If the resulting voltage is far from 0V, or HAL does not read the pin as FALSE, then your port cannot be used in *x* mode.

The external hardware that drives the control pins should also use open collector gates (e.g., 74LS05). On some computers, BIOS settings may affect whether *x* mode can be used. *SPP* mode is most likely to work.

No other combinations are supported, and a port cannot be changed from input to output once the driver is installed.

The `parport` driver can control up to 8 ports (defined by `MAX_PORTS` in `hal_parport.c`). The ports are numbered starting at zero.

12.1.1 Loading

The `hal_parport` driver is a real time component so it must be loaded into the real time thread with `loadrt`. The configuration string describes the parallel ports to be used, and (optionally) their types.

If the configuration string does not describe at least one port, it is an error.

```
loadrt hal_parport cfg="port [type] [port [type] ...]"
```

Specifying the Port Numbers below 16 refer to parallel ports detected by the system. This is the simplest way to configure the `hal_parport` driver and cooperates with the Linux `parport_pc` driver if it is loaded. A port of 0 is the first parallel port detected on the system, 1 is the next and so on.

Basic configuration This will use the first parallel port Linux detects:

```
loadrt hal_parport cfg="0"
```

Using the Port Address Instead, the port address may be specified using the hex notation with the `0x` prefix.

The config string represents the hexadecimal address of the port, optionally followed by a direction, all repeated for each port. The directions are *in*, *out*, or *x*, and determine the direction of the physical pins 2 to 9 of the D-Sub 25 connector. If the direction is not specified, the data group will by default be configured as outputs. For example:

```
loadrt hal_parport cfg="0x278 0x378 in 0x20A0 out"
```

This example installs the drivers for a port 0x0278, with pins 2 to 9 as outputs (by default, since neither *in* nor *out* is specified), a port 0x0378, with pins 2 to 9 as inputs and a 0x20A0 port, with pins 2 to 9 explicitly specified as outputs. Note that you must know the base address of the parallel ports to configure the drivers correctly. For ISA bus ports, this is usually not a problem, since the ports are almost always at a well-known address, such as 0x278 or 0x378 which are typically configured in the BIOS. The addresses of PCI bus cards are usually found with `lspci -v` in an *I/O ports* line, or in a kernel message after running `sudo modprobe -a parport_pc`. There is no default address, so if `<config-string>` does not contain at least one address, it is an error.

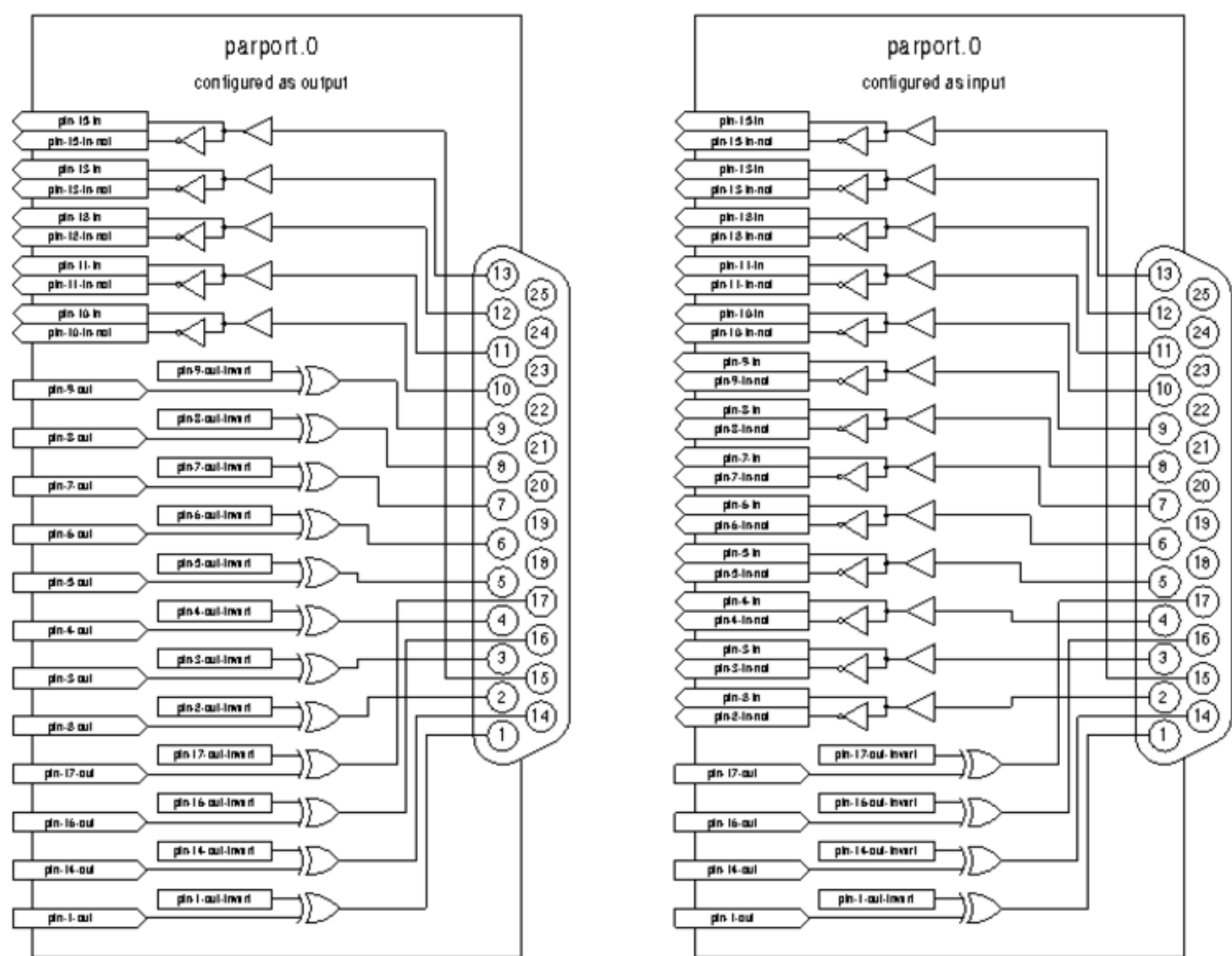


Figure 12.1: Parport block diagram

Type For each parallel port handled by the `hal_parport` driver, a *type* can optionally be specified. The type is one of *in*, *out*, *epi*, or *x*.

Table 12.1: Parallel Port Direction

Pin	in	out/epi	x
1	out	out	in
2	in	out	out
3	in	out	out
4	in	out	out
5	in	out	out
6	in	out	out
7	in	out	out
8	in	out	out
9	in	out	out
10	in	in	in
11	in	in	in
12	in	in	in
13	in	in	in

Table 12.1: (continued)

Pin	in	out/epp	x
14	out	out	in
15	in	in	in
16	out	out	in
17	out	out	in

If the type is not specified, the default is *out*.

A type of *epp* is the same as *out*, but the `hal_parport` driver requests that the port switch into EPP mode. The `hal_parport` driver does **not** use the EPP bus protocol, but on some systems EPP mode changes the electrical characteristics of the port in a way that may make some marginal hardware work better. The Gecko G540's charge pump is known to require this on some parallel ports.

See the Note above about mode *x*.

Example with two parallel ports This will enable two system-detected parallel ports, the first in output mode and the second in input mode:

```
loadrt hal_parport cfg="0 out 1 in"
```

Parport R/W Functions You must also direct LinuxCNC to run the *read* and *write* functions.

```
addf parport.0.read base-thread
addf parport.0.write base-thread
```

12.1.2 PCI Port Address

One good PCI parport card is made with the Netmos 9815 chipset. It has good +5V signals, and can come in a single or dual ports.

To find the I/O addresses for PCI cards open a terminal window and use the `list pci` command:

```
lspci -v
```

Look for the entry with "Netmos" in it. Example of a 2-port card:

```
0000:01:0a.0 Communication controller: \
    Netmos Technology PCI 9815 Multi-I/O Controller (rev 01)
Subsystem: LSI Logic / Symbios Logic 2POS (2 port parallel adapter)
Flags: medium devsel, IRQ 5
I/O ports at b800 [size=8]
I/O ports at bc00 [size=8]
I/O ports at c000 [size=8]
I/O ports at c400 [size=8]
I/O ports at c800 [size=8]
I/O ports at cc00 [size=16]
```

From experimentation, I've found the first port (the on-card port) uses the third address listed (c000), and the second port (the one that attaches with a ribbon cable) uses the first address listed (b800). The following example shows the onboard parallel port and a PCI parallel port using the default out direction.

```
loadrt hal_parport cfg="0x378 0xc000"
```

Please note that your values will differ. The Netmos cards are Plug-N-Play, and might change their settings depending on which slot you put them into, so if you like to 'get under the hood' and re-arrange things, be sure to check these values before you start LinuxCNC.

12.1.3 Pins

- *parport.<p>.pin-<n>-out* (bit) Drives a physical output pin.
- *parport.<p>.pin-<n>-in* (bit) Tracks a physical input pin.
- *parport.<p>.pin-<n>-in-not* (bit) Tracks a physical input pin, but inverted.

For each pin, *<p>* is the port number, and *<n>* is the physical pin number in the 25 pin D-shell connector.

For each physical output pin, the driver creates a single HAL pin, for example: *parport.0.pin-14-out*.

For each physical input pin, the driver creates two HAL pins, for example: *parport.0.pin-12-in* and *parport.0.pin-12-in-not*.

The *-in* HAL pin is TRUE if the physical pin is high, and FALSE if the physical pin is low. The *-in-not* HAL pin is inverted and is FALSE if the physical pin is high.

12.1.4 Parameters

- *parport.<p>.pin-<n>-out-invert* (bit) Inverts an output pin.
- *parport.<p>.pin-<n>-out-reset* (bit) (only for *out* pins) TRUE if this pin should be reset when the *-reset* function is executed.
- *parport.<p>.reset-time* (U32) The time (in nanoseconds) between a pin is set by *write* and reset by the *reset* function if it is enabled.

The *-invert* parameter determines whether an output pin is active high or active low. If *-invert* is FALSE, setting the HAL *-out* pin TRUE drives the physical pin high, and FALSE drives it low. If *-invert* is TRUE, then setting the HAL *-out* pin TRUE will drive the physical pin low.

12.1.5 Functions

- *parport.<p>.read* (funct) Reads physical input pins of port *<portnum>* and updates HAL *-in* and *-in-not* pins.
- *parport.read-all* (funct) Reads physical input pins of all ports and updates HAL *-in* and *-in-not* pins.
- *parport.<p>.write* (funct) Reads HAL *-out* pins of port *<p>* and updates that port's physical output pins.
- *parport.write-all* (funct) Reads HAL *-out* pins of all ports and updates all physical output pins.
- *parport.<p>.reset* (funct) Waits until *reset-time* has elapsed since the associated *write*, then resets pins to values indicated by *-out-invert* and *-in-not-invert* settings. *reset* must be later in the same thread as *write*. 'If' *-reset* is TRUE, then the *reset* function will set the pin to the value of *-out-invert*. This can be used in conjunction with stepgen's *doublefreq* to produce one step per period. The [stepgen stepspace](#) for that pin must be set to 0 to enable *doublefreq*.

The individual functions are provided for situations where one port needs to be updated in a very fast thread, but other ports can be updated in a slower thread to save CPU time. It is probably not a good idea to use both an *-all* function and an individual function at the same time.

12.1.6 Common problems

If loading the module reports

```
insmod: error inserting '/home/jepler/emc2/rtlib/hal_parport.ko':  
-1 Device or resource busy
```

then ensure that the standard kernel module *parport_pc* is not loaded ¹ and that no other device in the system has claimed the I/O ports.

If the module loads but does not appear to function, then the port address is incorrect.

12.1.7 Using DoubleStep

To setup DoubleStep on the parallel port you must add the function *parport.n.reset* after *parport.n.write* and configure *stepspace* to 0 and the reset time wanted. So that step can be asserted on every period in HAL and then toggled off by *parport* after being asserted for time specified by *parport.n.reset-time*.

For example:

```
loadrt hal_parport cfg="0x378 out"  
setp parport.0.reset-time 5000  
loadrt stepgen step_type=0,0,0  
addf parport.0.read base-thread  
addf stepgen.make-pulses base-thread  
addf parport.0.write base-thread  
addf parport.0.reset base-thread  
addf stepgen.capture-position servo-thread  
...  
setp stepgen.0.steplen 1  
setp stepgen.0.stepspace 0
```

More information on DoubleStep can be found on the [wiki](#).

12.1.8 probe_parport

In today's PCs, parallel ports may require a plug and play (PNP) configuration before they can be used. The kernel module *probe_parport* configures all PNP ports present. It must be loaded before *hal_parport*. On machines without a PNP port, it can be loaded but will have no effect.

12.1.8.1 Installing probe_parport

If, when *parport_pc* kernel module is loaded with command:

```
sudo modprobe -a parport_pc; sudo rmmod parport_pc
```

Linux kernel outputs a message similar to:

```
parport: PnPBIOS parport detected.
```

Then use of this module will probably be necessary.

Finally, HAL *parport* components should be loaded:

```
loadrt probe_parport  
loadrt hal_parport ...
```

¹In the LinuxCNC packages for Ubuntu, the file */etc/modprobe.d/emc2* generally prevents *parport_pc* from being automatically loaded.

12.2 AX5214H Driver

The Axiom Measurement & Control AX5214H is a 48 channel digital I/O board. It plugs into an ISA bus, and resembles a pair of 8255 chips. In fact it may be a pair of 8255 chips, but I'm not sure. If/when someone starts a driver for an 8255 they should look at the ax5214 code, much of the work is already done.

12.2.1 Installing

```
loadrt hal_ax5214h cfg="<config-string>"
```

The config string consists of a hex port address, followed by an 8 character string of "I" and "O" which sets groups of pins as inputs and outputs. The first two character set the direction of the first two 8 bit blocks of pins (0-7 and 8-15). The next two set blocks of 4 pins (16-19 and 20-23). The pattern then repeats, two more blocks of 8 bits (24-31 and 32-39) and two blocks of 4 bits (40-43 and 44-47). If more than one board is installed, the data for the second board follows the first. As an example, the string "0x220 IIIIOHOO 0x300 OIOOIOIO" installs drivers for two boards. The first board is at address 0x220, and has 36 inputs (0-19 and 24-39) and 12 outputs (20-23 and 40-47). The second board is at address 0x300, and has 20 inputs (8-15, 24-31, and 40-43) and 28 outputs (0-7, 16-23, 32-39, and 44-47). Up to 8 boards may be used in one system.

12.2.2 Pins

- (bit) *ax5214.<boardnum>.out-<pinnum>* — Drives a physical output pin.
- (bit) *ax5214.<boardnum>.in-<pinnum>* — Tracks a physical input pin.
- (bit) *ax5214.<boardnum>.in-<pinnum>-not* — Tracks a physical input pin, inverted.

For each pin, <boardnum> is the board number (starts at zero), and <pinnum> is the I/O channel number (0 to 47).

Note that the driver assumes active LOW signals. This is so that modules such as OPTO-22 will work correctly (TRUE means output ON, or input energized). If the signals are being used directly without buffering or isolation the inversion needs to be accounted for. The in- HAL pin is TRUE if the physical pin is low (OPTO-22 module energized), and FALSE if the physical pin is high (OPTO-22 module off). The in-<pinnum>-not HAL pin is inverted — it is FALSE if the physical pin is low (OPTO-22 module energized). By connecting a signal to one or the other, the user can determine the state of the input.

12.2.3 Parameters

- (bit) *ax5214.<boardnum>.out-<pinnum>-invert* — Inverts an output pin.

The -invert parameter determines whether an output pin is active high or active low. If -invert is FALSE, setting the HAL out- pin TRUE drives the physical pin low, turning ON an attached OPTO-22 module, and FALSE drives it high, turning OFF the OPTO-22 module. If -invert is TRUE, then setting the HAL out- pin TRUE will drive the physical pin high and turn the module OFF.

12.2.4 Functions

- (funct) *ax5214.<boardnum>.read* — Reads all digital inputs on one board.
- (funct) *ax5214.<boardnum>.write* — Writes all digital outputs on one board.

12.3 GS2 VFD Driver

This is a userspace HAL program for the GS2 series of VFD's at Automation Direct. ²

This component is loaded using the halcmd "loadusr" command:

```
loadusr -Wn spindle-vfd gs2_vfd -n spindle-vfd
```

The above command says: loadusr, wait for named to load, component gs2_vfd, named spindle-vfd. The HAL loadusr command is described in the [loadusr](#) chapter.

12.3.1 Command Line Options

- *-b* or *--bits* *<n>* (default: 8) Set number of data bits to *<n>*, where n must be from 5 to 8 inclusive.
- *-d* or *--device* *<path>* (default: /dev/ttyS0) Set the file path to the serial device node to use.
- *-g* or *--debug* Turn on debugging messages. This will also set the verbose flag. Debug mode will cause all modbus messages to be printed in hex on the terminal.
- *-n* or *--name* *<string>* (default: gs2_vfd) Set the name of the HAL module. The HAL comp name will be set to *<string>*, and all pin and parameter names will begin with *<string>*.
- *-p* or *--parity* *{even,odd,none}* (default: odd) Set serial parity to even, odd, or none.
- *-r* or *--rate* *<n>* (default: 38400) Set baud rate to *<n>*. It is an error if the rate is not one of the following: 110, 300, 600, 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200.
- *-s* or *--stopbits* *{1,2}* (default: 1) Set serial stop bits to 1 or 2
- *-t* or *--target* *<n>* (default: 1) Set MODBUS target (slave) number. This must match the device number you set on the GS2.
- *-v* or *--verbose* Turn on debug messages.
- *-A* or *--accel-seconds* *<n>* (default: 10.0) Seconds to accelerate the spindle from 0 to max. RPM.
- *-D* or *--decel-seconds* *<n>* (default: 0.0) Seconds to decelerate the spindle from max. RPM to 0. If set to 0.0 the spindle will be allowed to coast to a stop without controlled deceleration.
- *-R* or *--braking-resistor* This argument should be used when a braking resistor is installed on the GS2 VFD (see Appendix A of the GS2 manual). It disables deceleration over-voltage stall prevention (see GS2 modbus Parameter 6.05), allowing the VFD to keep braking even in situations where the motor is regenerating high voltage. The regenerated voltage gets safely dumped into the braking resistor.

Note

That if there are serial configuration errors, turning on verbose may result in a flood of timeout errors.

²In Europe the equivalent can be found under the brand name Omron.

12.3.2 Pins

Where `<n>` is `gs2_vfd` or the name given during loading with the `-n` option.

- `<n>.DC-bus-volts` (float, out) DC bus voltage of the VFD
- `<n>.at-speed` (bit, out) when drive is at commanded speed
- `<n>.err-reset` (bit, in) reset errors sent to VFD
- `<n>.firmware-revision` (s32, out) from the VFD
- `<n>.frequency-command` (float, out) from the VFD
- `<n>.frequency-out` (float, out) from the VFD
- `<n>.is-stopped` (bit, out) when the VFD reports 0 Hz output
- `<n>.load-percentage` (float, out) from the VFD
- `<n>.motor-RPM` (float, out) from the VFD
- `<n>.output-current` (float, out) from the VFD
- `<n>.output-voltage` (float, out) from the VFD
- `<n>.power-factor` (float, out) from the VFD
- `<n>.scale-frequency` (float, out) from the VFD
- `<n>.speed-command` (float, in) speed sent to VFD in RPM It is an error to send a speed faster than the Motor Max RPM as set in the VFD.
- `<n>.spindle-fwd` (bit, in) 1 for FWD and 0 for REV sent to VFD
- `<n>.spindle-rev` (bit, in) 1 for REV and 0 if off
- `<n>.spindle-on` (bit, in) 1 for ON and 0 for OFF sent to VFD
- `<n>.status-1` (s32, out) Drive Status of the VFD (see the GS2 manual)
- `<n>.status-2` (s32, out) Drive Status of the VFD (see the GS2 manual)

Note

The status value is a sum of all the bits that are on. So a 163 which means the drive is in the run mode is the sum of 3 (run) + 32 (freq set by serial) + 128 (operation set by serial).

12.3.3 Parameters

Where `<n>` is `gs2_vfd` or the name given during loading with the `-n` option.

- `<n>.error-count` (s32, RW)
- `<n>.loop-time` (float, RW) how often the modbus is polled (default: 0.1)
- `<n>.nameplate-HZ` (float, RW) Nameplate Hz of motor (default: 60)
- `<n>.nameplate-RPM` (float, RW) Nameplate RPM of motor (default: 1730)
- `<n>.retval` (s32, RW) the return value of an error in HAL
- `<n>.tolerance` (s32, RW) speed tolerance (default: 0.01)
- `<n>.ack-delay` (s32, RW) number of read/write cycles before checking at-speed (default 2)

For an example of using this component to drive a spindle see the [GS2 Spindle](#) example.

12.4 Mesa HostMot2 Driver

12.4.1 Introduction

HostMot2 is an FPGA configuration developed by Mesa Electronics for their line of *Anything I/O* motion control cards. The firmware is open source, portable and flexible. It can be configured (at compile-time) with zero or more instances (an object created at runtime) of each of several Modules: encoders (quadrature counters), PWM generators, and step/dir generators. The firmware can be configured (at run-time) to connect each of these instances to pins on the I/O headers. I/O pins not driven by a Module instance revert to general-purpose bi-directional digital I/O.

12.4.2 Firmware Binaries

50 Pin Header FPGA cards Several pre-compiled HostMot2 firmware binaries are available for the different Anything I/O boards. (This list is incomplete, check the `hostmot2-firmware` distribution for up-to-date firmware lists.)

- 3x20 (144 I/O pins): using `hm2_pci` module
 - 24-channel servo
 - 16-channel servo plus 24 step/dir generators
- 5i22 (96 I/O pins): using `hm2_pci` module
 - 16-channel servo
 - 8-channel servo plus 24 step/dir generators
- 5i20, 5i23, 4i65, 4i68 (72 I/O pins): using `hm2_pci` module
 - 12-channel servo
 - 8-channel servo plus 4 step/dir generators
 - 4-channel servo plus 8 step/dir generators
- 7i43 (48 I/O pins): using `hm2_7i43` module
 - 8-channel servo (8 PWM generators & 8 encoders)
 - 4-channel servo plus 4 step/dir generators

DB25 FPGA cards The 5i25 Superport FPGA card is preprogrammed when purchased and does not need a firmware binary.

12.4.3 Installing Firmware

Depending on how you installed LinuxCNC you may have to open the Synaptic Package Manager from the System menu and install the package for your Mesa card. The quickest way to find them is to do a search for *hostmot2* in the Synaptic Package Manager. Mark the firmware for installation, then apply.

12.4.4 Loading HostMot2

The LinuxCNC support for the HostMot2 firmware is split into a generic driver called *hostmot2* and two low-level I/O drivers for the Anything I/O boards. The low-level I/O drivers are *hm2_7i43* and *hm2_pci* (for all the PCI- and PC-104/Plus-based Anything I/O boards). The *hostmot2* driver must be loaded first, using a HAL command like this:

```
loadrt hostmot2
```

See the *hostmot2(9)* man page for details.

The *hostmot2* driver by itself does nothing, it needs access to actual boards running the HostMot2 firmware. The low-level I/O drivers provide this access. The low-level I/O drivers are loaded with commands like this:

```
loadrt hm2_pci config="firmware=hm2/5i20/SVST8_4.BIT  
num_encoders=3 num_pwmgens=3 num_stepgens=1"
```

The config parameters are described in the *hostmot2* man page.

12.4.5 Watchdog

The HostMot2 firmware may include a watchdog Module; if it does, the *hostmot2* driver will use it.

The watchdog must be petted by LinuxCNC periodically or it will bite. The *hm2* write function (see below) pets the watchdog.

When the watchdog bites, all the board's I/O pins are disconnected from their Module instances and become high-impedance inputs (pulled high). The state of the HostMot2 firmware modules is not disturbed (except for the configuration of the I/O Pins). Encoder instances keep counting quadrature pulses, and pwm- and step-generators keep generating signals (which are not relayed to the motors, because the I/O Pins have become inputs).

Resetting the watchdog resets the I/O pins to the configuration chosen at load-time.

If the firmware includes a watchdog, the following HAL objects will be exported:

12.4.5.1 Pins:

- *has_bit* - (bit i/o) True if the watchdog has bit, False if the watchdog has not bit. If the watchdog has bit and the *has_bit* bit is True, the user can reset it to False to resume operation.

12.4.5.2 Parameters:

- *timeout_ns* - (u32 read/write) Watchdog timeout, in nanoseconds. This is initialized to 5,000,000 (5 milliseconds) at module load time. If more than this amount of time passes between calls to the *hm2* write function, the watchdog will bite.

12.4.6 HostMot2 Functions

- *hm2_<BoardType>.<BoardNum>.read* - Read all inputs, update input HAL pins.
 - *hm2_<BoardType>.<BoardNum>.write* - Write all outputs.
 - *hm2_<BoardType>.<BoardNum>.read_gpio* - Read the GPIO input pins only. (This function is not available on the 7i43 due to limitations of the EPP bus.)
-

- `hm2_<BoardType>.<BoardNum>.write_gpio` - Write the GPIO control registers and output pins only. (This function is not available on the 7i43 due to limitations of the EPP bus.)

Note

The above `read_gpio` and `write_gpio` functions should not normally be needed, since the GPIO bits are read and written along with everything else in the standard `read` and `write` functions above, which are normally run in the servo thread.

The `read_gpio` and `write_gpio` functions were provided in case some very fast (frequently updated) I/O is needed. These functions should be run in the base thread. If you have need for this, please send an email and tell us about it, and what your application is.

12.4.7 Pinouts

The hostmot2 driver does not have a particular pinout. The pinout comes from the firmware that the hostmot2 driver sends to the Anything I/O board. Each firmware has different pinout, and the pinout depends on how many of the available encoders, pwmgens, and stepgens are used. To get a pinout list for your configuration after loading LinuxCNC in the terminal window type:

```
dmesg > hm2.txt
```

The resulting text file will contain lots of information as well as the pinout for the HostMot2 and any error and warning messages.

To reduce the clutter by clearing the message buffer before loading LinuxCNC type the following in the terminal window:

```
sudo dmesg -c
```

Now when you run LinuxCNC and then do a `dmesg > hm2.txt` in the terminal only the info from the time you loaded LinuxCNC will be in your file along with your pinout. The file will be in the current directory of the terminal window. Each line will contain the card name, the card number, the I/O Pin number, the connector and pin, and the usage. From this printout you will know the physical connections to your card based on your configuration.

An example of a 5i20 configuration:

```
[HOSTMOT2]
DRIVER=hm2_pci
BOARD=5i20
CONFIG="firmware=hm2/5i20/SVST8_4.BIT num_encoders=1 num_pwmgens=1 num_stepgens=3"
```

The above configuration produced this printout.

```
[ 1141.053386] hm2/hm2_5i20.0: 72 I/O Pins used:
[ 1141.053394] hm2/hm2_5i20.0: IO Pin 000 (P2-01): IOPort
[ 1141.053397] hm2/hm2_5i20.0: IO Pin 001 (P2-03): IOPort
[ 1141.053401] hm2/hm2_5i20.0: IO Pin 002 (P2-05): Encoder #0, pin B (Input)
[ 1141.053405] hm2/hm2_5i20.0: IO Pin 003 (P2-07): Encoder #0, pin A (Input)
[ 1141.053408] hm2/hm2_5i20.0: IO Pin 004 (P2-09): IOPort
[ 1141.053411] hm2/hm2_5i20.0: IO Pin 005 (P2-11): Encoder #0, pin Index (Input)
[ 1141.053415] hm2/hm2_5i20.0: IO Pin 006 (P2-13): IOPort
[ 1141.053418] hm2/hm2_5i20.0: IO Pin 007 (P2-15): PWMGen #0, pin Out0 (PWM or Up) (Output)
[ 1141.053422] hm2/hm2_5i20.0: IO Pin 008 (P2-17): IOPort
[ 1141.053425] hm2/hm2_5i20.0: IO Pin 009 (P2-19): PWMGen #0, pin Out1 (Dir or Down) (↔
    Output)
[ 1141.053429] hm2/hm2_5i20.0: IO Pin 010 (P2-21): IOPort
[ 1141.053432] hm2/hm2_5i20.0: IO Pin 011 (P2-23): PWMGen #0, pin Not-Enable (Output)
<snip>...
```

```
[ 1141.053589] hm2/hm2_5i20.0: IO Pin 060 (P4-25): StepGen #2, pin Step (Output)
[ 1141.053593] hm2/hm2_5i20.0: IO Pin 061 (P4-27): StepGen #2, pin Direction (Output)
[ 1141.053597] hm2/hm2_5i20.0: IO Pin 062 (P4-29): StepGen #2, pin (unused) (Output)
[ 1141.053601] hm2/hm2_5i20.0: IO Pin 063 (P4-31): StepGen #2, pin (unused) (Output)
[ 1141.053605] hm2/hm2_5i20.0: IO Pin 064 (P4-33): StepGen #2, pin (unused) (Output)
[ 1141.053609] hm2/hm2_5i20.0: IO Pin 065 (P4-35): StepGen #2, pin (unused) (Output)
[ 1141.053613] hm2/hm2_5i20.0: IO Pin 066 (P4-37): IOPort
[ 1141.053616] hm2/hm2_5i20.0: IO Pin 067 (P4-39): IOPort
[ 1141.053619] hm2/hm2_5i20.0: IO Pin 068 (P4-41): IOPort
[ 1141.053621] hm2/hm2_5i20.0: IO Pin 069 (P4-43): IOPort
[ 1141.053624] hm2/hm2_5i20.0: IO Pin 070 (P4-45): IOPort
[ 1141.053627] hm2/hm2_5i20.0: IO Pin 071 (P4-47): IOPort
[ 1141.053811] hm2/hm2_5i20.0: registered
[ 1141.053815] hm2_5i20.0: initialized AnyIO board at 0000:02:02.0
```

Note

That the I/O Pin nnn will correspond to the pin number shown on the HAL Configuration screen for GPIOs. Some of the StepGen, Encoder and PWMGen will also show up as GPIOs in the HAL Configuration screen.

12.4.8 PIN Files

The default pinout is described in a .PIN file (human-readable text). When you install a firmware package the .PIN files are installed in

```
/usr/share/doc/hostmot2-firmware-<board>/
```

12.4.9 Firmware

The selected firmware (.BIT file) and configuration is uploaded from the PC motherboard to the Mesa mothercard on LinuxCNC startup. If you are using Run In Place, you must still install a hostmot2-firmware-<board> package. There is more information about firmware and configuration in the *Configurations* section.

12.4.10 HAL Pins

The HAL pins for each configuration can be seen by opening up *Show HAL Configuration* from the Machine menu. All the HAL pins and parameters can be found there. The following figure is of the 5i20 configuration used above.

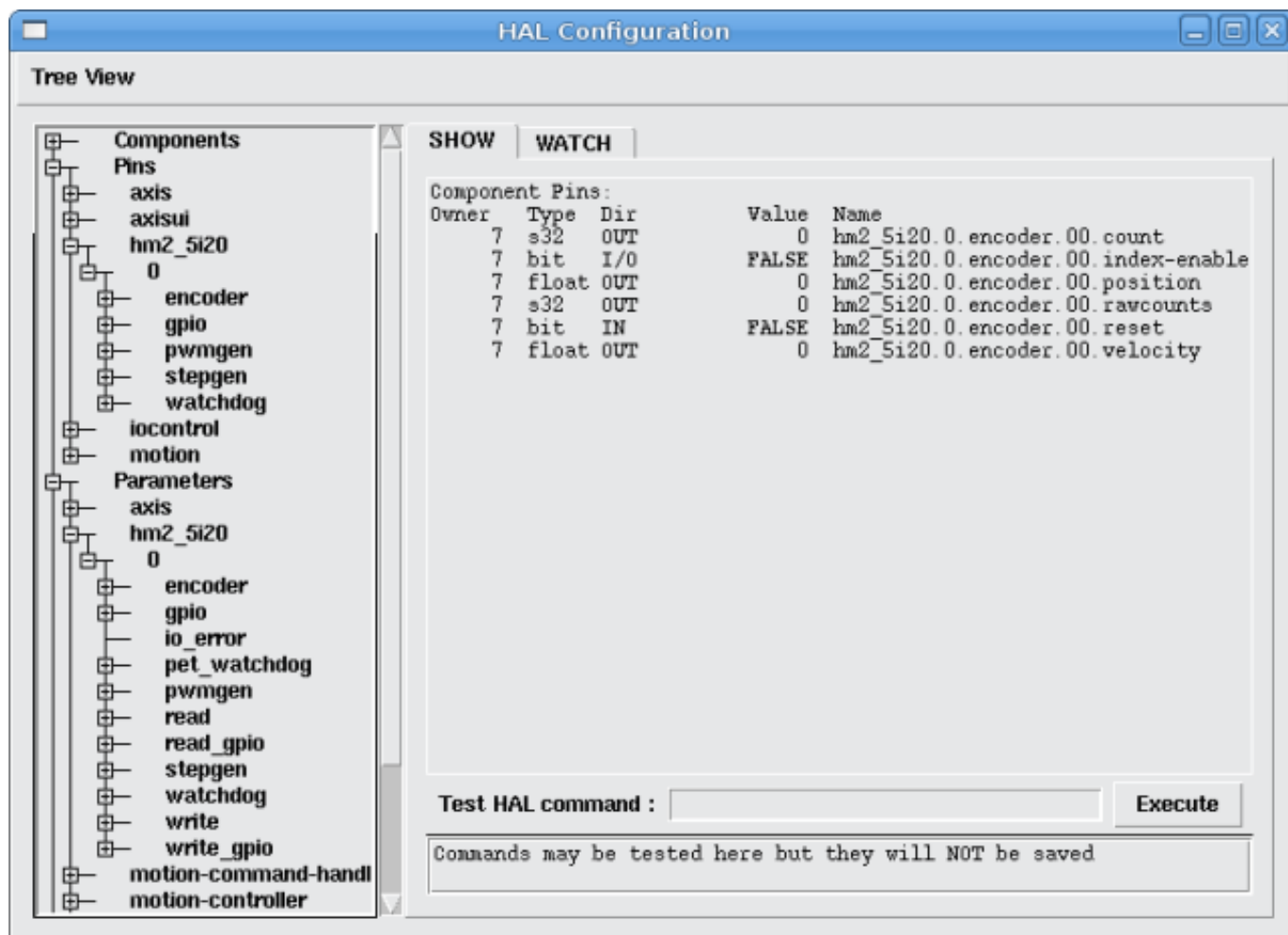


Figure 12.2: 5i20 HAL Pins

12.4.11 Configurations

The Hostmot2 firmware is available in several versions, depending on what you are trying to accomplish. You can get a reminder of what a particular firmware is for by looking at the name. Let's look at a couple of examples.

In the 7i43 (two ports), SV8 (*Servo 8*) would be for having 8 servos or fewer, using the *classic* 7i33 4-axis (per port) servo board. So 8 servos would use up all 48 signals in the two ports. But if you only needed 3 servos, you could say `num_encoders=3` and `num_pwmgens=3` and recover 5 servos at 6 signals each, thus gaining 30 bits of GPIO.

Or, in the 5i22 (four ports), SVST8_24 (*Servo 8, Stepper 24*) would be for having 8 servos or fewer (7i33 x2 again), and 24 steppers or fewer (7i47 x2). This would use up all four ports. If you only needed 4 servos you could say `num_encoders=4` and `num_pwmgens=4` and recover 1 port (and save a 7i33). And if you only needed 12 steppers you could say `num_stepgens=12` and free up one port (and save a 7i47). So in this way we can save two ports (48 bits) for GPIO.

Here are tables of the firmwares available in the official packages. There may be additional firmwares available at the Mesanet.com website that have not yet made it into the LinuxCNC official firmware packages, so check there too.

3x20 (6-port various) Default Configurations (The 3x20 comes in 1M, 1.5M, and 2M gate versions. So far, all firmware is available in all gate sizes.)

Firmware	Encoder	PWMGen	StepGen	GPIO
SV24	24	24	0	0
SVST16_24	16	16	24	0

5i22 (4-port PCI) Default Configurations (The 5i22 comes in 1M and 1.5M gate versions. So far, all firmware is available in all gate sizes.)

Firmware	Encoder	PWM	StepGen	GPIO
SV16	16	16	0	0
SVST2_4_7I47	4	2	4	72
SVST8_8	8	8	8	0
SVST8_24	8	8	24	0

5i23 (3-port PCI) Default Configurations (The 5i23 has 400k gates.)

Firmware	Encoder	PWM	StepGen	GPIO
SV12	12	12	0	0
SVST2_8	2	2	8 (tbl5)	12
SVST2_4_7I47	4	2	4	48
SV12_2X7I48_72	12	12	0	24
SV12IM_2X7I48_72 (+IM)	12	12	0	12
SVST4_8	4	4	8 (tbl5)	0
SVST8_4	8	8	4 (tbl5)	0
SVST8_4IM2	8 (+IM)	8	4	8
SVST8_8IM2	8 (+IM)	8	8	0
SVTP6_7I39	6	0 (6 BLDC)	0	0

5i20 (3-port PCI) Default Configurations (The 5i20 has 200k gates.)

Firmware	Encoder	PWM	StepGen	GPIO
SV12	12	12	0	0
SVST2_8	2	2	8 (tbl5)	12
SVST2_4_7I47	4	2	4	48
SV12_2X7I48_72	12	12	0	24
SV12IM_2X7I48_72 (+IM)	12	12	0	12
SVST8_4	8	8	4 (tbl5)	0
SVST8_4IM2	8 (+IM)	8	4	8

4i68 (3-port PC/104) Default Configurations (The 4i68 has 400k gates.)

Firmware	Encoder	PWM	StepGen	GPIO
SV12	12	12	0	0
SVST2_4_7I47	4	2	4	48
SVST4_8	4	4	8	0
SVST8_4	8	8	4	0
SVST8_4IM2	8 (+IM)	8	4	8
SVST8_8IM2	8 (+IM)	8	8	0

4i65 (3-port PC/104) Default Configurations (The 4i65 has 200k gates.)

Firmware	Encoder	PWM	StepGen	GPIO
SV12	12	12	0	0
SVST8_4	8	8	4	0

Firmware	Encoder	PWM	StepGen	GPIO
SVST8_4IM2	8 (+IM)	8	4	8

7i43 (2-port parallel) 400k gate versions, Default Configurations

Firmware	Encoder	PWM	StepGen	GPIO
SV8	8	8	0	0
SVST4_4	4	4	4 (tbl5)	0
SVST4_6	4	4	6 (tbl3)	0
SVST4_12	4	4	12	0
SVST2_4_7i47	4	2	4	24

7i43 (2-port parallel) 200k gate versions, Default Configurations

Firmware	Encoder	PWM	StepGen	GPIO
SV8	8	8	0	0
SVST4_4	4	4	4 (tbl5)	0
SVST4_6	4	4	6 (tbl3)	0
SVST2_4_7i47	4	2	4	24

Even though several cards may have the same named .BIT file you cannot use a .BIT file that is not for that card. Different cards have different clock frequencies so make sure you load the proper .BIT file for your card. Custom hm2 firmwares can be created for special applications and you may see some custom hm2 firmwares in the directories with the default ones.

When you load the board-driver (hm2_pci or hm2_7i43), you can tell it to disable instances of the three primary modules (pwmggen, stepgen, and encoder) by setting the count lower. Any I/O pins belonging to disabled module instances become GPIOs.

12.4.12 GPIO

General Purpose I/O pins on the board which are not used by a module instance are exported to HAL as *full* GPIO pins. Full GPIO pins can be configured at run-time to be inputs, outputs, or open drains, and have a HAL interface that exposes this flexibility. I/O pins that are owned by an active module instance are constrained by the requirements of the owning module, and have a restricted HAL interface.

GPIOs have names like *hm2_<BoardType>.<BoardNum>.gpio.<IONum>*. IONum. is a three-digit number. The mapping from IONum to connector and pin-on-that-connector is written to the syslog when the driver loads, and it's documented in Mesa's manual for the Anything I/O boards.

The hm2 GPIO representation is modeled after the Digital Inputs and Digital Outputs described in the Canonical Device Interface (part of the HAL General Reference document).

GPIO pins default to input.

12.4.12.1 Pins

- *in* - (Bit, Out) Normal state of the hardware input pin. Both full GPIO pins and I/O pins used as inputs by active module instances have this pin.
- *in_not* - (Bit, Out) Inverted state of the hardware input pin. Both full GPIO pins and I/O pins used as inputs by active module instances have this pin.
- *out* - (Bit, In) Value to be written (possibly inverted) to the hardware output pin. Only full GPIO pins have this pin.

12.4.12.2 Parameters

- *invert_output* - (Bit, RW) This parameter only has an effect if the *is_output* parameter is true. If this parameter is true, the output value of the GPIO will be the inverse of the value on the *out* HAL pin. Only full GPIO pins and I/O pins used as outputs by active module instances have this parameter. To invert an active module pin you have to invert the GPIO pin not the module pin.
- *is_opendrain* - (Bit, RW) This parameter only has an effect if the *is_output* parameter is true. If this parameter is false, the GPIO behaves as a normal output pin: the I/O pin on the connector is driven to the value specified by the *out* HAL pin (possibly inverted), and the value of the *in* and *in_not* HAL pins is undefined. If this parameter is true, the GPIO behaves as an open-drain pin. Writing 0 to the *out* HAL pin drives the I/O pin low, writing 1 to the *out* HAL pin puts the I/O pin in a high-impedance state. In this high-impedance state the I/O pin floats (weakly pulled high), and other devices can drive the value; the resulting value on the I/O pin is available on the *in* and *in_not* pins. Only full GPIO pins and I/O pins used as outputs by active module instances have this parameter.
- *is_output* - (Bit, RW) If set to 0, the GPIO is an input. The I/O pin is put in a high-impedance state (weakly pulled high), to be driven by other devices. The logic value on the I/O pin is available in the *in* and *in_not* HAL pins. Writes to the *out* HAL pin have no effect. If this parameter is set to 1, the GPIO is an output; its behavior then depends on the *is_opendrain* parameter. Only full GPIO pins have this parameter.

12.4.13 StepGen

StepGens have names like *hm2_<BoardType>.<BoardNum>.stepgen.<Instance>*. *Instance* is a two-digit number that corresponds to the HostMot2 stepgen instance number. There are *num_stepgens* instances, starting with 00.

Each stepgen allocates 2-6 I/O pins (selected at firmware compile time), but currently only uses two: Step and Direction outputs. ³

The StepGen representation is modeled on the stepgen software component. StepGen default is active high step output (high during step time low during step space). To invert a StepGen output pin you invert the corresponding GPIO pin that is being used by StepGen. To find the GPIO pin being used for the StepGen output run *dmesg* as shown above.

Each StepGen instance has the following pins and parameters:

12.4.13.1 Pins

- *control-type* - (Bit, In) Switches between position control mode (0) and velocity control mode (1). Defaults to position control (0).
- *counts* - (s32, Out) Feedback position in counts (number of steps).
- *enable* - (Bit, In) Enables output steps. When false, no steps are generated.
- *position-cmd* - (Float, In) Target position of stepper motion, in user-defined position units.
- *position-fb* - (Float, Out) Feedback position in user-defined position units (counts / *position_scale*).
- *velocity-cmd* - (Float, In) Target velocity of stepper motion, in user-defined position units per second. This pin is only used when the stepgen is in velocity control mode (*control-type*=1).
- *velocity-fb* - (Float, Out) Feedback velocity in user-defined position units per second.

³At present, the firmware supports multi-phase stepper outputs, but the driver doesn't. Interested volunteers are solicited.

12.4.13.2 Parameters

- *dirhold* - (u32, RW) Minimum duration of stable Direction signal after a step ends, in nanoseconds.
- *dirsetup* - (u32, RW) Minimum duration of stable Direction signal before a step begins, in nanoseconds.
- *maxaccel* - (Float, RW) Maximum acceleration, in position units per second per second. If set to 0, the driver will not limit its acceleration.
- *maxvel* - (Float, RW) Maximum speed, in position units per second. If set to 0, the driver will choose the maximum velocity based on the values of *steplen* and *stepspace* (at the time that *maxvel* was set to 0).
- *position-scale* - (Float, RW) Converts from counts to position units. $\text{position} = \text{counts} / \text{position_scale}$
- *step_type* - (u32, RW) Output format, like the *step_type* modparam to the software *stegen(9)* component. 0 = Step/Dir, 1 = Up/Down, 2 = Quadrature. In Quadrature mode (*step_type*=2), the *stepgen* outputs one complete Gray cycle (00 -> 01 -> 11 -> 10 -> 00) for each *step* it takes.
- *steplen* - (u32, RW) Duration of the step signal, in nanoseconds.
- *stepspace* - (u32, RW) Minimum interval between step signals, in nanoseconds.

12.4.13.3 Output Parameters

The Step and Direction pins of each StepGen have two additional parameters. To find which I/O pin belongs to which step and direction output run *dmesg* as described above.

- *invert_output* - (Bit, RW) This parameter only has an effect if the *is_output* parameter is true. If this parameter is true, the output value of the GPIO will be the inverse of the value on the *out* HAL pin.
- *is_opendrain* - (Bit, RW) If this parameter is false, the GPIO behaves as a normal output pin: the I/O pin on the connector is driven to the value specified by the *out* HAL pin (possibly inverted). If this parameter is true, the GPIO behaves as an open-drain pin. Writing 0 to the *out* HAL pin drives the I/O pin low, writing 1 to the *out* HAL pin puts the I/O pin in a high-impedance state. In this high-impedance state the I/O pin floats (weakly pulled high), and other devices can drive the value; the resulting value on the I/O pin is available on the *in* and *in_not* pins. Only full GPIO pins and I/O pins used as outputs by active module instances have this parameter.

12.4.14 PWMGen

PWMgens have names like *hm2_<BoardType>.<BoardNum>.pwmgen.<Instance>*. *Instance* is a two-digit number that corresponds to the HostMot2 *pwmgen* instance number. There are *num_pwmgens* instances, starting with 00.

In HM2, each *pwmgen* uses three output I/O pins: Not-Enable, Out0, and Out1. To invert a PWMGen output pin you invert the corresponding GPIO pin that is being used by PWMGen. To find the GPIO pin being used for the PWMGen output run *dmesg* as shown above.

The function of the Out0 and Out1 I/O pins varies with output-type parameter (see below).

The *hm2* *pwmgen* representation is similar to the software *pwmgen* component. Each *pwmgen* instance has the following pins and parameters:

12.4.14.1 Pins

- *enable* - (Bit, In) If true, the *pwmgen* will set its Not-Enable pin false and output its pulses. If *enable* is false, *pwmgen* will set its Not-Enable pin true and not output any signals.
- *value* - (Float, In) The current *pwmgen* command value, in arbitrary units.

12.4.14.2 Parameters

- *output-type* - (s32, RW) This emulates the *output_type* load-time argument to the software *pwmgen* component. This parameter may be changed at runtime, but most of the time you probably want to set it at startup and then leave it alone. Accepted values are 1 (PWM on Out0 and Direction on Out1), 2 (Up on Out0 and Down on Out1), 3 (PDM mode, PDM on Out0 and Dir on Out1), and 4 (Direction on Out0 and PWM on Out1, *for locked antiphase*).
- *scale* - (Float, RW) Scaling factor to convert *value* from arbitrary units to duty cycle: $dc = value / scale$. Duty cycle has an effective range of -1.0 to +1.0 inclusive, anything outside that range gets clipped.
- *pdm_frequency* - (u32, RW) This specifies the PDM frequency, in Hz, of all the *pwmgen* instances running in PDM mode (mode 3). This is the *pulse slot frequency*; the frequency at which the *pdm* generator in the Anything I/O board chooses whether to emit a pulse or a space. Each pulse (and space) in the PDM pulse train has a duration of $1/pdm_frequency$ seconds. For example, setting the *pdm_frequency* to 2e6 (2 MHz) and the duty cycle to 50% results in a 1 MHz square wave, identical to a 1 MHz PWM signal with 50% duty cycle. The effective range of this parameter is from about 1525 Hz up to just under 100 MHz. Note that the max frequency is determined by the *ClockHigh* frequency of the Anything I/O board; the 5i20 and 7i43 both have a 100 MHz clock, resulting in a 100 MHz max PDM frequency. Other boards may have different clocks, resulting in different max PDM frequencies. If the user attempts to set the frequency too high, it will be clipped to the max supported frequency of the board.
- *pwm_frequency* - (u32, RW) This specifies the PWM frequency, in Hz, of all the *pwmgen* instances running in the PWM modes (modes 1 and 2). This is the frequency of the variable-duty-cycle wave. Its effective range is from 1 Hz up to 193 kHz. Note that the max frequency is determined by the *ClockHigh* frequency of the Anything I/O board; the 5i20 and 7i43 both have a 100 MHz clock, resulting in a 193 kHz max PWM frequency. Other boards may have different clocks, resulting in different max PWM frequencies. If the user attempts to set the frequency too high, it will be clipped to the max supported frequency of the board. Frequencies below about 5 Hz are not terribly accurate, but above 5 Hz they're pretty close.

12.4.14.3 Output Parameters

The output pins of each PWMGen have two additional parameters. To find which I/O pin belongs to which output run *dmesg* as described above.

- *invert_output* - (Bit, RW) This parameter only has an effect if the *is_output* parameter is true. If this parameter is true, the output value of the GPIO will be the inverse of the value on the *out* HAL pin.
- *is_opendrain* - (Bit, RW) If this parameter is false, the GPIO behaves as a normal output pin: the I/O pin on the connector is driven to the value specified by the *out* HAL pin (possibly inverted). If this parameter is true, the GPIO behaves as an open-drain pin. Writing 0 to the *out* HAL pin drives the I/O pin low, writing 1 to the *out* HAL pin puts the I/O pin in a high-impedance state. In this high-impedance state the I/O pin floats (weakly pulled high), and other devices can drive the value; the resulting value on the I/O pin is available on the *in* and *in_not* pins. Only full GPIO pins and I/O pins used as outputs by active module instances have this parameter.

12.4.15 Encoder

Encoders have names like *hm2_<BoardType>.<BoardNum>.encoder.<Instance>..* *Instance* is a two-digit number that corresponds to the HostMot2 encoder instance number. There are *num_encoders* instances, starting with 00.

Each encoder uses three or four input I/O pins, depending on how the firmware was compiled. Three-pin encoders use A, B, and Index (sometimes also known as Z). Four-pin encoders use A, B, Index, and Index-mask.

The hm2 encoder representation is similar to the one described by the Canonical Device Interface (in the HAL General Reference document), and to the software encoder component. Each encoder instance has the following pins and parameters:

12.4.15.1 Pins

- *count* - (s32, Out) Number of encoder counts since the previous reset.
- *index-enable* - (Bit, I/O) When this pin is set to True, the count (and therefore also position) are reset to zero on the next Index (Phase-Z) pulse. At the same time, index-enable is reset to zero to indicate that the pulse has occurred.
- *position* - (Float, Out) Encoder position in position units (count / scale).
- *rawcounts* - (s32, Out) Total number of encoder counts since the start, not adjusted for index or reset.
- *reset* - (Bit, In) When this pin is TRUE, the count and position pins are set to 0. The value of the velocity pin is not affected by this. The driver does not reset this pin to FALSE after resetting the count to 0, that is the user's job.
- *velocity* - (Float, Out) Estimated encoder velocity in position units per second.

12.4.15.2 Parameters

- *counter-mode* - (Bit, RW) Set to False (the default) for Quadrature. Set to True for Up/Down or for single input on Phase A. Can be used for a frequency to velocity converter with a single input on Phase A when set to true.
- *filter* - (Bit, RW) If set to True (the default), the quadrature counter needs 15 clocks to register a change on any of the three input lines (any pulse shorter than this is rejected as noise). If set to False, the quadrature counter needs only 3 clocks to register a change. The encoder sample clock runs at 33 MHz on the PCI Anything I/O cards and 50 MHz on the 7i43.
- *index-invert* - (Bit, RW) If set to True, the rising edge of the Index input pin triggers the Index event (if index-enable is True). If set to False, the falling edge triggers.
- *index-mask* - (Bit, RW) If set to True, the Index input pin only has an effect if the Index-Mask input pin is True (or False, depending on the index-mask-invert pin below).
- *index-mask-invert* - (Bit, RW) If set to True, Index-Mask must be False for Index to have an effect. If set to False, the Index-Mask pin must be True.
- *scale* - (Float, RW) Converts from *count* units to *position* units. A quadrature encoder will normally have 4 counts per pulse so a 100 PPR encoder would be 400 counts per revolution. In *counter-mode* a 100 PPR encoder would have 100 counts per revolution as it only uses the rising edge of A and direction is B.
- *vel-timeout* - (Float, RW) When the encoder is moving slower than one pulse for each time that the driver reads the count from the FPGA (in the hm2_read() function), the velocity is harder to estimate. The driver can wait several iterations for the next pulse to arrive, all the while reporting the upper bound of the encoder velocity, which can be accurately guessed. This parameter specifies how long to wait for the next pulse, before reporting the encoder stopped. This parameter is in seconds.

12.4.16 5i25 Configuration

12.4.16.1 Firmware

The 5i25 firmware comes preloaded for the daughter card it is purchased with. So the *firmware=xxx.BIT* is not part of the hm2_pci configuration string when using a 5i25.

12.4.16.2 Configuration

Example configurations of the 5i25/7i76 and 5i25/7i77 cards are included in the [Configuration Selector](#).

If you like to roll your own configuration the following examples show how to load the drivers in the HAL file.

5i25 + 7i76 Card

```
# load the generic driver
loadrt hostmot2

# load the PCI driver and configure
loadrt hm2_pci config="num_encoders=1 num_stepgens=5 sserial_port_0=0XXX"
```

5i25 + 7i77 Card

```
# load the generic driver
loadrt hostmot2

# load the PCI driver and configure
loadrt hm2_pci config="num_encoders=6 num_pwmgens=6 sserial_port_0=0XXX"
```

12.4.16.3 SSERIAL Configuration

The *sserial_port_0=0XXX* configuration string sets some options for the smart serial daughter card. These options are specific for each daughter card. See the Mesa manual for more information on the exact usage.

12.4.16.4 7i77 Limits

The minlimit and maxlimit are bounds on the pin value (in this case the analog out value) fullscalemax is the scale factor.

These are by default set to the analog in or analog range (most likely in volts).

So for example on the 7i77 +-10V analog outputs, the default values are:

```
minlimit -10 maxlimit +10 maxfullscale 10
```

If you wanted to say scale the analog out of a channel to IPS for a velocity mode servo (say 24 IPS max) you could set the limits like this:

```
minlimit -24 maxlimit +24 maxfullscale 24
```

If you wanted to scale the analog out of a channel to RPM for a 0 to 6000 RPM spindle with 0-10V control you could set the limits like this:

```
minlimit 0 maxlimit 6000 maxfullscale 6000 (this would prevent unwanted negative output voltages from being set)
```

12.4.17 Example Configurations

Several example configurations for Mesa hardware are included with LinuxCNC. The configurations are located in the hm2-servo and hm2-stepper sections of the [Configuration Selector](#). Typically you will need the board installed for the configuration you pick to load. The examples are a good place to start and will save you time. Just pick the proper example from the LinuxCNC Configuration Selector and save a copy to your computer so you can edit it. To see the exact pins and parameters that your configuration gave you, open the Show HAL Configuration window from the Machine menu, or do *dmesg* as outlined above.

12.5 Motenc Driver

Vital Systems Motenc-100 and Motenc-LITE

The Vital Systems Motenc-100 and Motenc-LITE are 8- and 4-channel servo control boards. The Motenc-100 provides 8 quadrature encoder counters, 8 analog inputs, 8 analog outputs, 64 (68?) digital inputs, and 32 digital outputs. The Motenc-LITE has only 4 encoder counters, 32 digital inputs and 16 digital outputs, but it still has 8 analog inputs and 8 analog outputs. The driver automatically identifies the installed board and exports the appropriate HAL objects.

Installing:

```
loadrt hal_motenc
```

During loading (or attempted loading) the driver prints some useful debugging messages to the kernel log, which can be viewed with *dmesg*.

Up to 4 boards may be used in one system.

12.5.1 Pins

In the following pins, parameters, and functions, *<board>* is the board ID. According to the naming conventions the first board should always have an ID of zero. However this driver sets the ID based on a pair of jumpers on the board, so it may be non-zero even if there is only one board.

- *(s32) motenc.<board>.enc-<channel>-count* - Encoder position, in counts.
- *(float) motenc.<board>.enc-<channel>-position* - Encoder position, in user units.
- *(bit) motenc.<board>.enc-<channel>-index* - Current status of index pulse input.
- *(bit) motenc.<board>.enc-<channel>-idx-latch* - Driver sets this pin true when it latches an index pulse (enabled by latch-index). Cleared by clearing latch-index.
- *(bit) motenc.<board>.enc-<channel>-latch-index* - If this pin is true, the driver will reset the counter on the next index pulse.
- *(bit) motenc.<board>.enc-<channel>-reset-count* - If this pin is true, the counter will immediately be reset to zero, and the pin will be cleared.
- *(float) motenc.<board>.dac-<channel>-value* - Analog output value for DAC (in user units, see -gain and -offset)
- *(float) motenc.<board>.adc-<channel>-value* - Analog input value read by ADC (in user units, see -gain and -offset)
- *(bit) motenc.<board>.in-<channel>* - State of digital input pin, see canonical digital input.

- (bit) *motenc.<board>.in-<channel>-not* - Inverted state of digital input pin, see canonical digital input.
- (bit) *motenc.<board>.out-<channel>* - Value to be written to digital output, see canonical digital output.
- (bit) *motenc.<board>.estop-in* - Dedicated estop input, more details needed.
- (bit) *motenc.<board>.estop-in-not* - Inverted state of dedicated estop input.
- (bit) *motenc.<board>.watchdog-reset* - Bidirectional, - Set TRUE to reset watchdog once, is automatically cleared.

12.5.2 Parameters

- (float) *motenc.<board>.enc-<channel>-scale* - The number of counts / user unit (to convert from counts to units).
- (float) *motenc.<board>.dac-<channel>-offset* - Sets the DAC offset.
- (float) *motenc.<board>.dac-<channel>-gain* - Sets the DAC gain (scaling).
- (float) *motenc.<board>.adc-<channel>-offset* - Sets the ADC offset.
- (float) *motenc.<board>.adc-<channel>-gain* - Sets the ADC gain (scaling).
- (bit) *motenc.<board>.out-<channel>-invert* - Inverts a digital output, see canonical digital output.
- (u32) *motenc.<board>.watchdog-control* - Configures the watchdog.
The value may be a bitwise OR of the following values:

Bit #	Value	Meaning
0	1	Timeout is 16ms if set, 8ms if unset
1	2	
2	4	Watchdog is enabled
3	8	
4	16	Watchdog is automatically reset by DAC writes (the HAL dac-write function)

Typically, the useful values are 0 (watchdog disabled) or 20 (8ms watchdog enabled, cleared by dac-write).

- (u32) *motenc.<board>.led-view* - Maps some of the I/O to onboard LEDs.

12.5.3 Functions

- (funct) *motenc.<board>.encoder-read* - Reads all encoder counters.
- (funct) *motenc.<board>.adc-read* - Reads the analog-to-digital converters.
- (funct) *motenc.<board>.digital-in-read* - Reads digital inputs.
- (funct) *motenc.<board>.dac-write* - Writes the voltages to the DACs.
- (funct) *motenc.<board>.digital-out-write* - Writes digital outputs.
- (funct) *motenc.<board>.misc-update* - Updates misc stuff.

12.6 MB2HAL

12.6.1 Introduction

MB2HAL is a generic userspace HAL component to communicate with one or more Modbus devices. So far, there are two options to communicate with a Modbus device:

1. One option is to create a HAL component as a driver see [VFD Modbus](#).
2. Another option is to use Classic Ladder which has Modbus built in see << cha:classicladder,ClassicLadder >>.
3. Now there is a third option that consists of a "generic" driver configured by text file, this is called MB2HAL.

Why MB2HAL? Consider using Mb2hal if:

- You have to write a new driver and you don't know anything about programming.
- You need to use Classic Ladder "only" to manage the Modbus connections.
- You have to discover and configure first time the Modbus transactions. Mb2hal have debug levels to facilitate the low level protocol debug.
- You have more than one device to connect. Mb2hal is very efficient managing multiple devices, transactions and links. Currently I am monitoring two axis drivers using a Rs232 port, a VFD driver using another Rs232 port, and a remote I/O using TCP/IP.
- You want a protocol to connect your Arduino to HAL. Look the included sample configuration file, sketch and library for Arduino Modbus.

12.6.2 Usage

- a. Create a config file from the example below
 1. Set component name (optional)
Set HAL_MODULE_NAME=mymodule (default HAL_MODULE_NAME=mb2hal)
 2. Load the modbus HAL userspace component
- b. Default component name: `loadusr -W mb2hal config=config_file.ini`
- c. Custom component name: `loadusr -Wn mymodule mb2hal config=config_file.ini`

12.6.3 Options

12.6.3.1 Init Section

[MB2HAL_INIT]

Value	Type	Required	Description
INIT_DEBUG	Integer	No	Debug level of init and INI file parsing. 0 = silent 1 = error messages (default) 2 = OK confirmation messages 3 = debugging messages 4 = maximum debugging messages (only in transactions)
HAL_MODULE_NAME	String	No	HAL module (component) name. Defaults to "mb2hal".
SLOWDOWN	Float	No	Insert a delay of "FLOAT seconds" between transactions in order to not to have a lot of logging and facilitate the debugging. Useful when using DEBUG=3 (NOT INIT_DEBUG=3). It affects ALL transactions. Use "0.0" for normal activity.
TOTAL_TRANSACTIONS	Integer	Yes	The number of total Modbus transactions. There is no maximum.

12.6.3.2 Transaction Sections

One transaction section is required per transaction, starting at [TRANSACTION_00] and counting up sequentially. If there is a new link (not transaction), you must provide the REQUIRED parameters 1st time. Warning: Any OPTIONAL parameter not specified are copied from the previous transaction.

Value	Type	Required	Description
LINK_TYPE	String	Yes	You must specify either a "serial" or "tcp" link for the first transaction. Later transactions will use the previous transaction link if not specified.
TCP_IP	IP address	If LINK_TYPE=tcp	The Modbus slave device IP address. Ignored if LINK_TYPE=serial.
TCP_PORT	Integer	No	The Modbus slave device TCP port. Defaults to 502. Ignored if LINK_TYPE=serial.
SERIAL_PORT	String	If LINK_TYPE=serial	The serial port. For example "/dev/ttyS0". Ignored if LINK_TYPE=tcp.
SERIAL_BAUD	Integer	If LINK_TYPE=serial	The baud rate. Ignored if LINK_TYPE=tcp.
SERIAL_BITS	Integer	If LINK_TYPE=serial	Data bits. One of 5, 6, 7, 8. Ignored if LINK_TYPE=tcp.
SERIAL_PARITY	String	If LINK_TYPE=serial	Data parity. One of: even, odd, none. Ignored if LINK_TYPE=tcp.
SERIAL_STOP	Integer	If LINK_TYPE=serial	Stop bits. One of 1, 2. Ignored if LINK_TYPE=tcp.
SERIAL_DELAY_MS	Integer	If LINK_TYPE=serial	Serial port delay between transactions of this section only. Ignored if LINK_TYPE=tcp.
MB_SLAVE_ID	Integer	Yes	Modbus slave number.
FIRST_ELEMENT	Integer	Yes	The first element address.
NELEMENTS	Integer	Unless PIN_NAMES is specified	The number of elements. It is an error to specify both NELEMENTS and PIN_NAMES. The pin names will be sequential numbers e.g. mb2hal.plcin.01
PIN_NAMES	List	Unless NELEMENTS is specified	A list of element names. These names will be used for the pin names, e.g. mb2hal.plcin.cycle_start. NOTE: There must be no white space characters in the list. Example: PIN_NAMES=cycle_start,stop,feed_hold

Value	Type	Required	Description
MB_TX_CODE	String	Yes	Modbus transaction function code (see specifications): <ul style="list-style-type: none"> • <code>fnct_01_read_coils</code> • <code>fnct_02_read_discrete_inputs</code> • <code>fnct_03_read_holding_registers</code> • <code>fnct_04_read_input_registers</code> • <code>fnct_05_write_single_coil</code> • <code>fnct_06_write_single_register</code> • <code>fnct_15_write_multiple_coils</code> • <code>fnct_16_write_multiple_registers</code>
MB_RESPONSE_TIMEOUT_MS	Integer	No	Response timeout for this transaction. In ms. Defaults to 500 ms. This is how much to wait for 1st byte before raise an error.
MB_BYTE_TIMEOUT_MS	Integer	No	Byte timeout for this transaction. In ms. Defaults to 500 ms. This is how much to wait from byte to byte before raise an error.
HAL_TX_NAME	String	No	Instead of giving the transaction number, use a name. Example: <code>mb2hal.00.01</code> could become <code>mb2hal.plcin.01</code> . The name must not exceed 28 characters. NOTE: when using names be careful that you don't end up with two transactions using the same name.
MAX_UPDATE_RATE	Float	No	Maximum update rate in Hz. Defaults to 0.0 (0.0 = as soon as available = infinite). NOTE: This is a maximum rate and the actual rate may be lower. If you want to calculate it in ms use $(1000 / \text{required_ms})$. Example: 100 ms = $\text{MAX_UPDATE_RATE}=10.0$, because $1000.0 \text{ ms} / 100.0 \text{ ms} = 10.0 \text{ Hz}$.
DEBUG	String	No	Debug level for this transaction only. See INIT_DEBUG parameter above.

12.6.3.3 Error codes

While debugging transactions, note the returned "ret[]" value correspond to:

Modbus protocol exceptions:

- 0x01 - ILLEGAL_FUNCTION - the FUNCTION code received in the query is not allowed or invalid.
- 0x02 - ILLEGAL_DATA_ADDRESS - the DATA ADDRESS received in the query is not an allowable address for the slave or is invalid.
- 0x03 - ILLEGAL_DATA_VALUE - a VALUE contained in the data query field is not an allowable value or is invalid.
- 0x04 - SLAVE_DEVICE_FAILURE - SLAVE (or MASTER) device unrecoverable FAILURE while attempting to perform the requested action.
- 0x04 - SERVER_FAILURE - (see above).
- 0x05 - ACKNOWLEDGE - This response is returned to PREVENT A TIMEOUT in the master. A long duration of time is required to process the request in the slave.
- 0x06 - SLAVE_DEVICE_BUSY - The slave (or server) is BUSY. Retransmit the request later.
- 0x06 - SERVER_BUSY - (see above).
- 0x07 - NEGATIVE_ACKNOWLEDGE - Unsuccessful programming request using function code 13 or 14.

- 0x08 - MEMORY_PARITY_ERROR - SLAVE parity error in MEMORY.
- 0x0A (-10) - GATEWAY_PROBLEM_PATH - Gateway path(s) not available.
- 0x0B (-11) - GATEWAY_PROBLEM_TARGET - The target device failed to respond (generated by master, not slave).

Program or connection:

- 0x0C (-12) - COMM_TIME_OUT
- 0x0D (-13) - PORT_SOCKET_FAILURE
- 0x0E (-14) - SELECT_FAILURE
- 0x0F (-15) - TOO_MANY_DATAS
- 0x10 (-16) - INVALID_CRC
- 0x11 (-17) - INVALID_EXCEPTION_CODE

12.6.4 Example config file

Click [here](#) to download.

```
#This .INI file is also the HELP, MANUAL and HOW-TO file for mb2hal.

#Load the modbus HAL userspace module as the examples below,
#change to match your own HAL_MODULE_NAME and .ini file name
#Using HAL_MODULE_NAME=mb2hal or nothing (default): loadusr -W mb2hal config=config_file. ↵
ini
#Using HAL_MODULE_NAME=mymodule: loadusr -Wn mymodule mb2hal config=config_file.ini

# ++++++
# Common section
# ++++++
[MB2HAL_INIT]

#OPTIONAL: Debug level of init and INI file parsing.
# 0 = silent.
# 1 = error messages (default).
# 2 = OK confirmation messages.
# 3 = debugging messages.
# 4 = maximum debugging messages (only in transactions).
INIT_DEBUG=3

#OPTIONAL: HAL module (component) name. Defaults to "mb2hal".
HAL_MODULE_NAME=mb2hal

#OPTIONAL: Insert a delay of "FLOAT seconds" between transactions in order
#to not to have a lot of logging and facilitate the debugging.
#Useful when using DEBUG=3 (NOT INIT_DEBUG=3)
#It affects ALL transactions.
#Use "0.0" for normal activity.
SLOWDOWN=0.0

#REQUIRED: The number of total Modbus transactions. There is no maximum.
TOTAL_TRANSACTIONS=9

# ++++++
# Transactions
```

```
# ++++++
#One transaction section is required per transaction, starting at 00 and counting up ↵
#sequentially.
#If there is a new link (not transaction), you must provide the REQUIRED parameters 1st ↵
#time.
#Warning: Any OPTIONAL parameter not specified are copied from the previous transaction.
[TRANSACTION_00]

#REQUIRED: You must specify either a "serial" or "tcp" link for the first transaction.
#Later transaction will use the previous transaction link if not specified.
LINK_TYPE=tcp

#if LINK_TYPE=tcp then REQUIRED (only 1st time): The Modbus slave device ip address.
#if LINK_TYPE=serial then IGNORED
TCP_IP=192.168.2.10

#if LINK_TYPE=tcp then OPTIONAL.
#if LINK_TYPE=serial then IGNORED
#The Modbus slave device tcp port. Defaults to 502.
TCP_PORT=502

#if LINK_TYPE=serial then REQUIRED (only 1st time).
#if LINK_TYPE=tcp then IGNORED
#The serial port.
SERIAL_PORT=/dev/ttyS0

#if LINK_TYPE=serial then REQUIRED (only 1st time).
#if LINK_TYPE=tcp then IGNORED
#The baud rate.
SERIAL_BAUD=115200

#if LINK_TYPE=serial then REQUIRED (only 1st time).
#if LINK_TYPE=tcp then IGNORED
#Data bits. One of 5,6,7,8.
SERIAL_BITS=8

#if LINK_TYPE=serial then REQUIRED (only 1st time).
#if LINK_TYPE=tcp then IGNORED
#Data parity. One of: even, odd, none.
SERIAL_PARITY=none

#if LINK_TYPE=serial then REQUIRED (only 1st time).
#if LINK_TYPE=tcp then IGNORED
#Stop bits. One of 1, 2.
SERIAL_STOP=2

#if LINK_TYPE=serial then OPTIONAL:
#if LINK_TYPE=tcp then IGNORED
#Serial port delay between for this transaction only.
#In ms. Defaults to 0.
SERIAL_DELAY_MS=10

#REQUIRED (only 1st time).
#Modbus slave number.
MB_SLAVE_ID=1

#REQUIRED: The first element address.
FIRST_ELEMENT=0

#REQUIRED unless PIN_NAMES is specified: The number of elements.
#It is an error to specify both NELEMENTS and PIN_NAMES
#The pin names will be sequential numbers e.g mb2hal.plcin.01
```

```
#NELEMENTS=4
```

```
#REQUIRED unless NELEMENTS is specified: A list of element names.
```

```
#these names will be used for the pin names, e.g mb2hal.plcin.cycle_start
```

```
#NOTE: there must be no white space characters in the list
```

```
PIN_NAMES=cycle_start,stop,feed_hold
```

```
#REQUIRED: Modbus transaction function code (see www.modbus.org specifications).
```

```
# fnct_01_read_coils (01 = 0x01)
```

```
# fnct_02_read_discrete_inputs (02 = 0x02)
```

```
# fnct_03_read_holding_registers (03 = 0x03)
```

```
# fnct_04_read_input_registers (04 = 0x04)
```

```
# fnct_05_write_single_coil (05 = 0x05)
```

```
# fnct_06_write_single_register (06 = 0x06)
```

```
# fnct_15_write_multiple_coils (15 = 0x0F)
```

```
# fnct_16_write_multiple_registers (16 = 0x10)
```

```
#
```

```
# Created pins:
```

```
# fnct_01_read_coils:
```

```
# fnct_02_read_discrete_inputs:
```

```
# mb2hal.m.n.bit (output)
```

```
# mb2hal.m.n.bit-inv (output)
```

```
# fnct_03_read_holding_registers:
```

```
# fnct_04_read_input_registers:
```

```
# mb2hal.m.n.float (output)
```

```
# mb2hal.m.n.int (output)
```

```
# fnct_05_write_single_coil:
```

```
# mb2hal.m.n.bit (input)
```

```
# NELEMENTS needs to be 1 or PIN_NAMES must contain just one name.
```

```
# fnct_06_write_single_register:
```

```
# mb2hal.m.n.float (input)
```

```
# mb2hal.m.n.int (input)
```

```
# NELEMENTS needs to be 1 or PIN_NAMES must contain just one name.
```

```
# Both pin values are added and limited to 65535 (UINT16_MAX). Normally use one and let ←  
the other open (read as 0).
```

```
# fnct_15_write_multiple_coils:
```

```
# mb2hal.m.n.bit (input)
```

```
# fnct_16_write_multiple_registers:
```

```
# mb2hal.m.n.float (input)
```

```
# mb2hal.m.n.int (input)
```

```
# Both pin values are added and limited to 65535 (UINT16_MAX). Normally use one and let ←  
the other open (read as 0).
```

```
#
```

```
# m = HAL_TX_NAME or transaction number if not set, n = element number (NELEMENTS) or name ←  
from PIN_NAMES
```

```
# Example: mb2hal.00.01.<type> (transaction=00, second register=01 (00 is the first one))
```

```
# mb2hal.TxName.01.<type> (HAL_TX_NAME=TxName, second register=01 (00 is the first ←  
one))
```

```
MB_TX_CODE=fnct_03_read_holding_registers
```

```
#OPTIONAL: Response timeout for this transaction. In INTEGER ms. Defaults to 500 ms.
```

```
#This is how much to wait for 1st byte before raise an error.
```

```
MB_RESPONSE_TIMEOUT_MS=500
```

```
#OPTIONAL: Byte timeout for this transaction. In INTEGER ms. Defaults to 500 ms.
```

```
#This is how much to wait from byte to byte before raise an error.
```

```
MB_BYTE_TIMEOUT_MS=500
```

```
#OPTIONAL: Instead of giving the transaction number, use a name.
```

```
#Example: mb2hal.00.01 could become mb2hal.plcin.01
```

```
#The name must not exceed 28 characters.
```

```
#NOTE: when using names be careful that you dont end up with two transactions
```

```

#using the same name.
HAL_TX_NAME=remoteIOcfg

#OPTIONAL: Maximum update rate in HZ. Defaults to 0.0 (0.0 = as soon as available = ↔
infinite).
#NOTE: This is a maximum rate and the actual rate may be lower.
#If you want to calculate it in ms use (1000 / required_ms).
#Example: 100 ms = MAX_UPDATE_RATE=10.0, because 1000.0 ms / 100.0 ms = 10.0 Hz
MAX_UPDATE_RATE=0.0

#OPTIONAL: Debug level for this transaction only.
#See INIT_DEBUG parameter above.
DEBUG=2

#While DEBUGGING transactions note the returned "ret[]" value correspond to:
#/* Modbus protocol exceptions */
#ILLEGAL_FUNCTION      -0x01 the FUNCTION code received in the query is not allowed or ↔
invalid.
#ILLEGAL_DATA_ADDRESS  -0x02 the DATA ADDRESS received in the query is not an allowable ↔
address for the slave or is invalid.
#ILLEGAL_DATA_VALUE    -0x03 a VALUE contained in the data query field is not an ↔
allowable value or is invalid.
#SLAVE_DEVICE_FAILURE  -0x04 SLAVE (or MASTER) device unrecoverable FAILURE while ↔
attempting to perform the requested action.
#SERVER_FAILURE        -0x04 (see above).
#ACKNOWLEDGE           -0x05 This response is returned to PREVENT A TIMEOUT in the master ↔
.
#                      A long duration of time is required to process the request ↔
in the slave.
#SLAVE_DEVICE_BUSY     -0x06 The slave (or server) is BUSY. Retrasmit the request later.
#SERVER_BUSY           -0x06 (see above).
#NEGATIVE_ACKNOWLEDGE  -0x07 Unsuccessful programming request using function code 13 or ↔
14.
#MEMORY_PARITY_ERROR   -0x08 SLAVE parity error in MEMORY.
#GATEWAY_PROBLEM_PATH  -0x0A (-10) Gateway path(s) not available.
#GATEWAY_PROBLEM_TARGET -0x0B (-11) The target device failed to repond (generated by ↔
master, not slave).
#/* Program or connection */
#COMM_TIME_OUT         -0x0C (-12)
#PORT_SOCKET_FAILURE   -0x0D (-13)
#SELECT_FAILURE        -0x0E (-14)
#TOO_MANY_DATAS        -0x0F (-15)
#INVALID_CRC           -0x10 (-16)
#INVALID_EXCEPTION_CODE -0x11 (-17)

[TRANSACTION_01]
MB_TX_CODE=fnct_01_read_coils
FIRST_ELEMENT=1024
NELEMENTS=24
HAL_TX_NAME=remoteIOin
MAX_UPDATE_RATE=0.0
DEBUG=1

[TRANSACTION_02]
MB_TX_CODE=fnct_02_read_discrete_inputs
FIRST_ELEMENT=1280
NELEMENTS=8
HAL_TX_NAME=readStatus
MAX_UPDATE_RATE=0.0

[TRANSACTION_03]
MB_TX_CODE=fnct_05_write_single_coil

```

```
FIRST_ELEMENT=100
NELEMENTS=1
HAL_TX_NAME=setEnableout
MAX_UPDATE_RATE=0.0

[TRANSACTION_04]
MB_TX_CODE=fnct_15_write_multiple_coils
FIRST_ELEMENT=150
NELEMENTS=10
HAL_TX_NAME=remoteIOout
MAX_UPDATE_RATE=0.0

[TRANSACTION_05]
LINK_TYPE=serial
SERIAL_PORT=/dev/ttyS0
SERIAL_BAUD=115200
SERIAL_BITS=8
SERIAL_PARITY=none
SERIAL_STOP=2
SERIAL_DELAY_MS=50
MB_SLAVE_ID=1
MB_TX_CODE=fnct_03_read_holding_registers
FIRST_ELEMENT=1
NELEMENTS=2
HAL_TX_NAME=XDrive01
MAX_UPDATE_RATE=0.0
DEBUG=1

[TRANSACTION_06]
MB_TX_CODE=fnct_04_read_input_registers
FIRST_ELEMENT=12
NELEMENTS=3
HAL_TX_NAME=XDrive02
MAX_UPDATE_RATE=10.0
DEBUG=1

[TRANSACTION_07]
MB_TX_CODE=fnct_06_write_single_register
FIRST_ELEMENT=20
NELEMENTS=1
HAL_TX_NAME=XDrive03
MAX_UPDATE_RATE=0.0
DEBUG=1

[TRANSACTION_08]
MB_TX_CODE=fnct_16_write_multiple_registers
FIRST_ELEMENT=55
NELEMENTS=8
HAL_TX_NAME=XDrive04
MAX_UPDATE_RATE=10.0
DEBUG=1
```

12.6.5 Pins

Note

Yellow = New in LinuxCNC 2.9

12.6.5.1 fnct_01_read_coils

- mb2hal.m.n.bit *bit out*
- mb2hal.m.n.bit-inv *bit out*

12.6.5.2 fnct_02_read_discrete_inputs

- mb2hal.m.n.bit *bit out*
- mb2hal.m.n.bit-inv *bit out*

12.6.5.3 fnct_03_read_holding_registers

- mb2hal.m.n.float *float out*
- mb2hal.m.n.int *s32 out*

12.6.5.4 fnct_04_read_input_registers

- mb2hal.m.n.float *float out*
- mb2hal.m.n.int *s32 out*

12.6.5.5 fnct_05_write_single_coil

- mb2hal.m.n.bit *bit in*

NELEMENTS needs to be 1 or PIN_NAMES must contain just one name.

12.6.5.6 fnct_06_write_single_register

- mb2hal.m.n.float *float in*
- mb2hal.m.n.int *s32 in*

NELEMENTS needs to be 1 or PIN_NAMES must contain just one name. Both pin values are added and limited to 65535 (UINT16_MAX). Use one and let the other open (read as 0).

12.6.5.7 fnct_15_write_multiple_coils

- mb2hal.m.n.bit *bit in*
-

12.6.5.8 fnct_16_write_multiple_registers

- `mb2hal.m.n.float float in`
- `mb2hal.m.n.int s32 in`

Both pin values are added and limited to 65535 (UINT16_MAX). Use one and let the other open (read as 0).

`m` = HAL_TX_NAME or transaction number if not set, `n` = element number (NELEMENTS) or name from PIN_NAMES

Example:

- `mb2hal.00.01.<type>` (transaction=00, second register=01 (00 is the first one))
- `mb2hal.TxName.01.<type>` (HAL_TX_NAME=TxName, second register=01 (00 is the first one))

12.7 Opto22 Driver

PCI AC5 ADAPTER CARD / HAL DRIVER

12.7.1 The Adapter Card

This is a card made by Opto22 for adapting the PCI port to solid state relay racks such as their standard or G4 series. It has 2 ports that can control up to 24 points each and has 4 on board LEDs. The ports use 50 pin connectors the same as Mesa boards. Any relay racks/breakout boards that work with Mesa Cards should work with this card with the understanding any encoder counters, PWM, etc., would have to be done in software. The AC5 does not have any *smart* logic on board, it is just an adapter.

See the manufacturer's website for more info:

http://www.opto22.com/site/pr_details.aspx?cid=4&item=PCI-AC5

I would like to thank Opto22 for releasing info in their manual, easing the writing of this driver!

12.7.2 The Driver

This driver is for the PCI AC5 card and will not work with the ISA AC5 card. The HAL driver is a realtime module. It will support 4 cards as is (more cards are possible with a change in the source code). Load the basic driver like so:

```
loadrt opto_ac5
```

This will load the driver which will search for max 4 boards. It will set I/O of each board's 2 ports to a default setting. The default configuration is for 12 inputs then 12 outputs. The pin name numbers correspond to the position on the relay rack. For example the pin names for the default I/O setting of port 0 would be:

- `opto_ac5.0.port0.in-00` - They would be numbered from 00 to 11
- `opto_ac5.0.port0.out-12` - They would be numbered 12 to 23 port 1 would be the same.

12.7.3 Pins

- *opto_ac5.[BOARDNUMBER].port[PORTNUMBER].in-[PINNUMBER] OUT bit* -
- *opto_ac5.[BOARDNUMBER].port[PORTNUMBER].in-[PINNUMBER]-not OUT bit* - Connect a HAL bit signal to this pin to read an I/O point from the card. The PINNUMBER represents the position in the relay rack. Eg. PINNUMBER 0 is position 0 in a Opto22 relay rack and would be pin 47 on the 50 pin header connector. The -not pin is inverted so that LOW gives TRUE and HIGH gives FALSE.
- *opto_ac5.[BOARDNUMBER].port[PORTNUMBER].out-[PINNUMBER] IN bit* - Connect a HAL bit signal to this pin to write to an I/O point of the card. The PINNUMBER represents the position in the relay rack. Eg. PINNUMBER 23 is position 23 in a Opto22 relay rack and would be pin 1 on the 50 pin header connector.
- *opto_ac5.[BOARDNUMBER].led[NUMBER] OUT bit* - Turns one of the 4 onboard LEDs on/off. LEDs are numbered 0 to 3.

BOARDNUMBER can be 0-3 PORTNUMBER can be 0 or 1. Port 0 is closest to the card bracket.

12.7.4 Parameters

- *opto_ac5.[BOARDNUMBER].port[PORTNUMBER].out-[PINNUMBER]-invert W bit* - When TRUE, invert the meaning of the corresponding -out pin so that TRUE gives LOW and FALSE gives HIGH.

12.7.5 FUNCTIONS

- *opto_ac5.0.digital-read* - Add this to a thread to read all the input points.
- *opto_ac5.0.digital-write* - Add this to a thread to write all the output points and LEDs.

For example the pin names for the default I/O setting of port 0 would be:

```
opto_ac5.0.port0.in-00
```

They would be numbered from 00 to 11

```
opto_ac5.0.port0.out-12
```

They would be numbered 12 to 23 port 1 would be the same.

12.7.6 Configuring I/O Ports

To change the default setting load the driver something like so:

```
loadrt opto_ac5 portconfig0=0xffff portconfig1=0xff0000
```

Of course changing the numbers to match the I/O you would like. Each port can be set up different.

Here's how to figure out the number: The configuration number represents a 32 bit long code to tell the card which I/O points are output vrs input. The lower 24 bits are the I/O points of one port. The 2 highest bits are for 2 of the on board LEDs. A one in any bit position makes the I/O point an output. The two highest bits must be output for the LEDs to work. The driver will automatically set the two highest bits for you, we won't talk about them.

The easiest way to do this is to fire up the calculator under APPLICATIONS/ACCESSORIES. Set it to scientific (click view). Set it BINARY (radio button Bin). Press 1 for every output you want and/or zero

for every input. Remember that HAL pin 00 corresponds to the rightmost bit. 24 numbers represent the 24 I/O points of one port. So for the default setting (12 inputs then 12 outputs) you would push 1 twelve times (that's the outputs) then 0 twelve times (that's the inputs). Notice the first I/O point is the lowest (rightmost) bit. (that bit corresponds to HAL pin 00 .looks backwards) You should have 24 digits on the screen. Now push the Hex radio button. The displayed number (fff000) is the configport number (put a 0x in front of it designating it as a HEX number).

Another example: To set the port for 8 outputs and 16 inputs (the same as a Mesa card). Here is the 24 bits represented in a BINARY number. Bit 1 is the rightmost number:

16 zeros for the 16 inputs and 8 ones for the 8 outputs

```
000000000000000011111111
```

This converts to FF on the calculator, so 0xff is the number to use for portconfig0 and/or portconfig1 when loading the driver.

12.7.7 Pin Numbering

HAL pin 00 corresponds to bit 1 (the rightmost) which represents position 0 on an Opto22 relay rack. HAL pin 01 corresponds to bit 2 (one spot to the left of the rightmost) which represents position 1 on an Opto22 relay rack. HAL pin 23 corresponds to bit 24 (the leftmost) which represents position 23 on an Opto22 relay rack.

HAL pin 00 connects to pin 47 on the 50 pin connector of each port. HAL pin 01 connects to pin 45 on the 50 pin connector of each port. HAL pin 23 connects to pin 1 on the 50 pin connector of each port.

Note that Opto22 and Mesa use opposite numbering systems: Opto22 position 23 = connector pin 1, and the position goes down as the connector pin number goes up. Mesa Hostmot2 position 1 = connector pin 1, and the position number goes up as the connector pin number goes up.

12.8 Pico Drivers

Pico Systems has a family of boards for doing analog servo, stepper, and PWM (digital) servo control. The boards connect to the PC through a parallel port working in EPP mode. Although most users connect one board to a parallel port, in theory any mix of up to 8 or 16 boards can be used on a single parport. One driver serves all types of boards. The final mix of I/O depends on the connected board(s). The driver doesn't distinguish between boards, it simply numbers I/O channels (encoders, etc) starting from 0 on the first board. The driver is named `hal_ppmc.ko`. The analog servo interface is also called the PPMC for Parallel Port Motion Control. There is also the Universal Stepper Controller, abbreviated the USC. And the Universal PWM Controller, or UPC.

Installing:

```
loadrt hal_ppmc port_addr=<addr1>[,<addr2>[,<addr3>...]]
```

The `port_addr` parameter tells the driver what parallel port(s) to check. By default, `<addr1>` is 0x0378, and `<addr2>` and following are not used. The driver searches the entire address space of the enhanced parallel port(s) at `port_addr`, looking for any board(s) in the PPMC family. It then exports HAL pins for whatever it finds. During loading (or attempted loading) the driver prints some useful debugging messages to the kernel log, which can be viewed with `dmesg`.

Up to 3 parport buses may be used, and each bus may have up to 8 (or possibly 16 PPMC) devices on it.

12.8.1 Command Line Options

There are several options that can be specified on the loadrt command line. First, the USC and UPC can have an 8-bit DAC added for spindle speed control and similar functions. This can be specified with the `extradac=0xnn[,0xmm]` parameter. The part enclosed in `[]` allows you to specify this option on more than one board of the system. The first hex digit tells which EPP bus is being referred to, it corresponds to the order of the port addresses in the `port_addr` parameter, where `<addr1>` would be zero here. So, for the first EPP bus, the first USC or UPC board would be described as `0x00`, the second USC or UPC on the same bus would be `0x02`. (Note that each USC or UPC takes up two addresses, so if one is at 00, the next would have to be 02.)

Alternatively, the 8 digital output pins can be used as additional digital outputs, it works the same way as above with the syntax : `extradout=0xnn'`. The `extradac` and `extradout` options are mutually exclusive on each board, you can only specify one.

The UPC and PPMC encoder boards can timestamp the arrival of encoder counts to refine the derivation of axis velocity. This derived velocity can be fed to the PID hal component to produce smoother D term response. The syntax is : `timestamp=0xnn[,0xmm]`, this works the same way as above to select which board is being configured. Default is to not enable the timestamp option. If you put this option on the command line, it enables the option. The first `n` selects the EPP bus, the second one matches the address of the board having the option enabled. The driver checks the revision level of the board to make sure it has firmware supporting the feature, and produces an error message if the board does not support it.

The PPMC encoder board has an option to select the encoder digital filter frequency. (The UPC has the same ability via DIP switches on the board.) Since the PPMC encoder board doesn't have these extra DIP switches, it needs to be selected via a command-line option. By default, the filter runs at 1 MHz, allowing encoders to be counted up to about 900 kHz (depending on noise and quadrature accuracy of the encoder.) The options are 1, 2.5, 5 and 10 MHz. These are set with a parameter of 1,2,5 and 10 (decimal) which is specified as the hex digit "A". These are specified in a manner similar to the above options, but with the frequency setting to the left of the bus/address digits. So, to set 5 MHz on the encoder board at address 3 on the first EPP bus, you would write : `enc_clock=0x503`

It was recently discovered that some parallel port chips would not work with the ppmc driver. Especially, the Oxford OXPIC952 chip on the SIIG PCIe parallel port cards had this trouble. The ppmc driver in all LinuxCNC versions starting from 2.7.8 have been corrected for this problem by default. However, this possibly could cause problems with really old EPP parallel port hardware, so there is a command line option to go back to the previous behavior. The new behavior is set by default, or by adding the parameter `epp_dir=0` on the command line. To get the old behavior, add `epp_dir=1` to the command line. All parallel ports I have here work with the new default behavior. As on the other parameters, it is possible to give a list, like `epp_dir=1,0,1` to set different settings for each of up to 3 parallel ports.

12.8.2 Pins

In the following pins, parameters, and functions, `<port>` is the parallel port ID. According to the naming conventions the first port should always have an ID of zero. All the boards have some method of setting the address on the EPP bus. USC and UPC have simple provisions for only two addresses, but jumper foil cuts allow up to 4 boards to be addressed. The PPMC boards have 16 possible addresses. In all cases, the driver enumerates the boards by type and exports the appropriate HAL pins. For instance, the encoders will be enumerated from zero up, in the same order as the address switches on the board specify. So, the first board will have encoders 0 — 3, the second board would have encoders 4 — 7. The first column after the bullet tells which boards will have this HAL pin or parameter associated with it. All means that this pin is available on all three board types. Option means that this pin will only be exported when that option is enabled by an optional parameter in the loadrt HAL command. These options require the board to have a sufficient revision level to support the feature.

- (All s32 output) `ppmc.<port>.encoder.<channel>.count` - Encoder position, in counts.

- (All s32 output) `ppmc.<port>.encoder.<channel>.delta` - Change in counts since last read, in raw encoder count units.
- (All float output) `'ppmc.<port>.encoder.<channel>.velocity` - Velocity scaled in user units per second. On PPMC and USC this is derived from raw encoder counts per servo period, and hence is affected by encoder granularity. On UPC boards with the 8/21/09 and later firmware, velocity estimation by timestamping encoder counts can be used to improve the smoothness of this velocity output. This can be fed to the PID HAL component to produce a more stable servo response. This function has to be enabled in the HAL command line that starts the PPMC driver, with the `timestamp=0x00` option.
- (All float output) `ppmc.<port>.encoder.<channel>.position` - Encoder position, in user units.
- (All bit bidir) `ppmc.<port>.encoder.<channel>.index-enable` - Connect to `joint.#.index-enable` for home-to-index. This is a bidirectional HAL signal. Setting it to true causes the encoder hardware to reset the count to zero on the next encoder index pulse. The driver will detect this and set the signal back to false.
- (PPMC float output) `ppmc.<port>.DAC.<channel>.value` - sends a signed value to the 16-bit Digital to Analog Converter on the PPMC DAC16 board commanding the analog output voltage of that DAC channel.
- (UPC bit input) `ppmc.<port>.pwm.<channel>.enable` - Enables a PWM generator.
- (UPC float input) `ppmc.<port>.pwm.<channel>.value` - Value which determines the duty cycle of the PWM waveforms. The value is divided by `pwm.<channel>.scale`, and if the result is 0.6 the duty cycle will be 60%, and so on. Negative values result in the duty cycle being based on the absolute value, and the direction pin is set to indicate negative.
- (USC bit input) `ppmc.<port>.stepgen.<channel>.enable` - Enables a step pulse generator.
- (USC float input) `ppmc.<port>.stepgen.<channel>.velocity` - Value which determines the step frequency. The value is multiplied by `stepgen.<channel>.scale`, and the result is the frequency in steps per second. Negative values result in the frequency being based on the absolute value, and the direction pin is set to indicate negative.
- (All bit output) `ppmc.<port>.din.<channel>.in` - State of digital input pin, see canonical digital input.
- (All bit output) `ppmc.<port>.din.<channel>.in-not` - Inverted state of digital input pin, see canonical digital input.
- (All bit input) `ppmc.<port>.dout.<channel>.out` - Value to be written to digital output, see canonical digital output.
- (Option float input) `ppmc.<port>.DAC8-<channel>.value` - Value to be written to analog output, range from 0 to 255. This sends 8 output bits to J8, which should have a Spindle DAC board connected to it. 0 corresponds to zero Volts, 255 corresponds to 10 Volts. The polarity of the output can be set for always minus, always plus, or can be controlled by the state of SSR1 (plus when on) and SSR2 (minus when on). You must specify `extradac = 0x00` on the HAL command line that loads the PPMC driver to enable this function on the first USC or UPC board.
- (Option bit input) `ppmc.<port>.dout.<channel>.out` - Value to be written to one of the 8 extra digital output pins on J8. You must specify `extradout = 0x00` on the HAL command line that loads the ppcm driver to enable this function on the first USC or UPC board. `extradac` and `extradout` are mutually exclusive features as they use the same signal lines for different purposes. These output pins will be enumerated after the standard digital outputs of the board.

12.8.3 Parameters

- (All float) `ppmc.<port>.encoder.<channel>.scale` - The number of counts / user unit (to convert from counts to units).
- (UPC float) `ppmc.<port>.pwm.<channel-range>.freq` - The PWM carrier frequency, in Hz. Applies to a group of four consecutive PWM generators, as indicated by `<channel-range>`. Minimum is 610 Hz, maximum is 500 kHz.
- (PPMC float) `ppmc.<port>.DAC.<channel>.scale` - Sets scale of DAC16 output channel such that an output value equal to the `1/scale` value will produce an output of + or - value Volts. So, if the scale parameter is 0.1 and you send a value of 0.5, the output will be 5.0 Volts.
- (UPC float) `ppmc.<port>.pwm.<channel>.scale` - Scaling for PWM generator. If `scale` is X, then the duty cycle will be 100% when the `value` pin is X (or -X).
- (UPC float) `ppmc.<port>.pwm.<channel>.max-dc` - Maximum duty cycle, from 0.0 to 1.0.
- (UPC float) `ppmc.<port>.pwm.<channel>.min-dc` - Minimum duty cycle, from 0.0 to 1.0.
- (UPC float) `ppmc.<port>.pwm.<channel>.duty-cycle` - Actual duty cycle (used mostly for troubleshooting.)
- (UPC bit) `ppmc.<port>.pwm.<channel>.bootstrap` - If true, the PWM generator will generate a short sequence of pulses of both polarities when E-stop goes false, to reset the shutdown latches on some PWM servo drives.
- (USC u32) `ppmc.<port>.stepgen.<channel-range>.setup-time` - Sets minimum time between direction change and step pulse, in units of 100 ns. Applies to a group of four consecutive step generators, as indicated by `<channel-range>`. Values between 200 ns and 25.5 μ s can be specified.
- (USC u32) `ppmc.<port>.stepgen.<channel-range>.pulse-width` - Sets width of step pulses, in units of 100 ns. Applies to a group of four consecutive step generators, as indicated by `<channel-range>`. Values between 200 ns and 25.5 μ s may be specified.
- (USC u32) `ppmc.<port>.stepgen.<channel-range>.pulse-space-min` - Sets minimum time between pulses, in units of 100 ns. Applies to a group of four consecutive step generators, as indicated by `<channel-range>`. Values between 200 ns and 25.5 μ s can be specified. The maximum step rate is:

$$\frac{1}{100\text{ns} * (\text{pulsewidth} + \text{pulsespacemin})}$$
- (USC float) `ppmc.<port>.stepgen.<channel>.scale` - Scaling for step pulse generator. The step frequency in Hz is the absolute value of `velocity * scale`.
- (USC float) `ppmc.<port>.stepgen.<channel>.max-vel` - The maximum value for `velocity`. Commands greater than `max-vel` will be clamped. Also applies to negative values. (The absolute value is clamped.)
- (USC float) `ppmc.<port>.stepgen.<channel>.frequency` - Actual step pulse frequency in Hz (used mostly for troubleshooting.)
- (Option float) `ppmc.<port>.DAC8.<channel>.scale` - Sets scale of extra DAC output such that an output value equal to `scale` gives a magnitude of 10.0 V output. (The sign of the output is set by jumpers and/or other digital outputs.)
- (Option bit) `ppmc.<port>.dout.<channel>.invert` - Inverts a digital output, see canonical digital output.
- (Option bit) `ppmc.<port>.dout.<channel>.invert` - Inverts a digital output pin of J8, see canonical digital output.

12.8.4 Functions

- (All funct) `ppmc.<port>.read` - Reads all inputs (digital inputs and encoder counters) on one port. These reads are organized into blocks of contiguous registers to be read in a block to minimize CPU overhead.
- (All funct) `ppmc.<port>.write` - Writes all outputs (digital outputs, steppens, PWMs) on one port. These writes are organized into blocks of contiguous registers to be written in a block to minimize CPU overhead.

12.9 Pluto P Driver

12.9.1 General Info

The Pluto-P is a FPGA board featuring the ACEX1K chip from Altera.

12.9.1.1 Requirements

1. A Pluto-P board
2. An EPP-compatible parallel port, configured for EPP mode in the system BIOS or a PCI EPP compatible parallel port card.

Note

The Pluto P board requires EPP mode. Netmos98xx chips do not work in EPP mode. The Pluto P board will work on some computers and not on others. There is no known pattern to which computers work and which don't work.

For more information on PCI EPP compatible parallel port cards see the [LinuxCNC Supported Hardware](#) page on the wiki.

12.9.1.2 Connectors

- The Pluto-P board is shipped with the left connector presoldered, with the key in the indicated position. The other connectors are unpopulated. There does not seem to be a standard 12-pin IDC connector, but some of the pins of a 16P connector can hang off the board next to QA3/QZ3.
- The bottom and right connectors are on the same .1" grid, but the left connector is not. If OUT2...OUT9 are not required, a single IDC connector can span the bottom connector and the bottom two rows of the right connector.

12.9.1.3 Physical Pins

- Read the ACEX1K datasheet for information about input and output voltage thresholds. The pins are all configured in *LVTTTL/LVCMOS* mode and are generally compatible with 5V TTL logic.
 - Before configuration and after properly exiting LinuxCNC, all Pluto-P pins are tristated with weak pull-ups (20 kΩ min, 50 kΩ max). If the watchdog timer is enabled (the default), these pins are also tristated after an interruption of communication between LinuxCNC and the board. The watchdog timer takes approximately 6.5 ms to activate. However, software bugs in the `pluto_servo` firmware or LinuxCNC can leave the Pluto-P pins in an undefined state.
-

- In pwm+dir mode, by default dir is HIGH for negative values and LOW for positive values. To select HIGH for positive values and LOW for negative values, set the corresponding dout-NN-invert parameter TRUE to invert the signal.
- The index input is triggered on the rising edge. Initial testing has shown that the QZx inputs are particularly noise sensitive, due to being polled every 25 ns. Digital filtering has been added to filter pulses shorter than 175 ns (seven polling times). Additional external filtering on all input pins, such as a Schmitt buffer or inverter, RC filter, or differential receiver (if applicable) is recommended.
- The IN1...IN7 pins have 22 Ω series resistors to their associated FPGA pins. No other pins have any sort of protection for out-of-spec voltages or currents. It is up to the integrator to add appropriate isolation and protection. Traditional parallel port optoisolator boards do not work with pluto_servo due to the bidirectional nature of the EPP protocol.

12.9.1.4 LED

- When the device is unprogrammed, the LED glows faintly. When the device is programmed, the LED glows according to the duty cycle of PWM0 ($LED = UP0 \text{ xor } DOWN0$) or STEPGEN0 ($LED = STEP0 \text{ xor } DIR0$).

12.9.1.5 Power

- A small amount of current may be drawn from VCC. The available current depends on the unregulated DC input to the board. Alternately, regulated +3.3VDC may be supplied to the FPGA through these VCC pins. The required current is not yet known, but is probably around 50mA plus I/O current.
- The regulator on the Pluto-P board is a low-dropout type. Supplying 5V at the power jack will allow the regulator to work properly.

12.9.1.6 PC interface

- Only a single pluto_servo or pluto_step board is supported.

12.9.1.7 Rebuilding the FPGA firmware

The `src/hal/drivers/pluto_servo_firmware/` and `src/hal/drivers/pluto_step_firmware/` subdirectories contain the Verilog source code plus additional files used by Quartus for the FPGA firmwares. Altera's Quartus II software is required to rebuild the FPGA firmware. To rebuild the firmware from the .hdl and other source files, open the .qpf file and press CTRL-L. Then, recompile LinuxCNC.

Like the HAL hardware driver, the FPGA firmware is licensed under the terms of the GNU General Public License.

The gratis version of Quartus II runs only on Microsoft Windows, although there is apparently a paid version that runs on Linux.

12.9.1.8 For more information

Some additional information about it is available from [KNJC LLC](#) and from [the developer's blog](#).

12.9.2 Pluto Servo

The `pluto_servo` system is suitable for control of a 4-axis CNC mill with servo motors, a 3-axis mill with PWM spindle control, a lathe with spindle encoder, etc. The large number of inputs allows a full set of limit switches.

This driver features:

- 4 quadrature channels with 40 MHz sample rate. The counters operate in 4x mode. The maximum useful quadrature rate is 8191 counts per LinuxCNC servo cycle, or about 8 MHz for LinuxCNC's default 1 ms servo rate.
- 4 PWM channels, *up/down* or *pwm+dir* style. 4095 duty cycles from -100% to +100%, including 0%. The PWM period is approximately 19.5 kHz (40 MHz / 2047). A PDM-like mode is also available.
- 18 digital outputs: 10 dedicated, 8 shared with PWM functions. (Example: A lathe with unidirectional PWM spindle control may use 13 total digital outputs)
- 20 digital inputs: 8 dedicated, 12 shared with Quadrature functions. (Example: A lathe with index pulse only on the spindle may use 13 total digital inputs.)
- EPP communication with the PC. The EPP communication typically takes around 100 µs on machines tested so far, enabling servo rates above 1 kHz.

12.9.2.1 Pinout

- *UPx* - The *up* (up/down mode) or *pwm* (pwm+direction mode) signal from PWM generator X. May be used as a digital output if the corresponding PWM channel is unused, or the output on the channel is always negative. The corresponding digital output invert may be set to TRUE to make UPx active low rather than active high.
 - *DNx* - The *down* (up/down mode) or *direction* (pwm+direction mode) signal from PWM generator X. May be used as a digital output if the corresponding PWM channel is unused, or the output on the channel is never negative. The corresponding digital output invert may be set to TRUE to make DNx active low rather than active high.
 - *QAx*, *QBx* - The A and B signals for Quadrature counter X. May be used as a digital input if the corresponding quadrature channel is unused.
 - *QZx* - The Z (index) signal for quadrature counter X. May be used as a digital input if the index feature of the corresponding quadrature channel is unused.
 - *INx* - Dedicated digital input #x
 - *OUTx* - Dedicated digital output #x
 - *GND* - Ground
 - *VCC* - +3.3V regulated DC
-

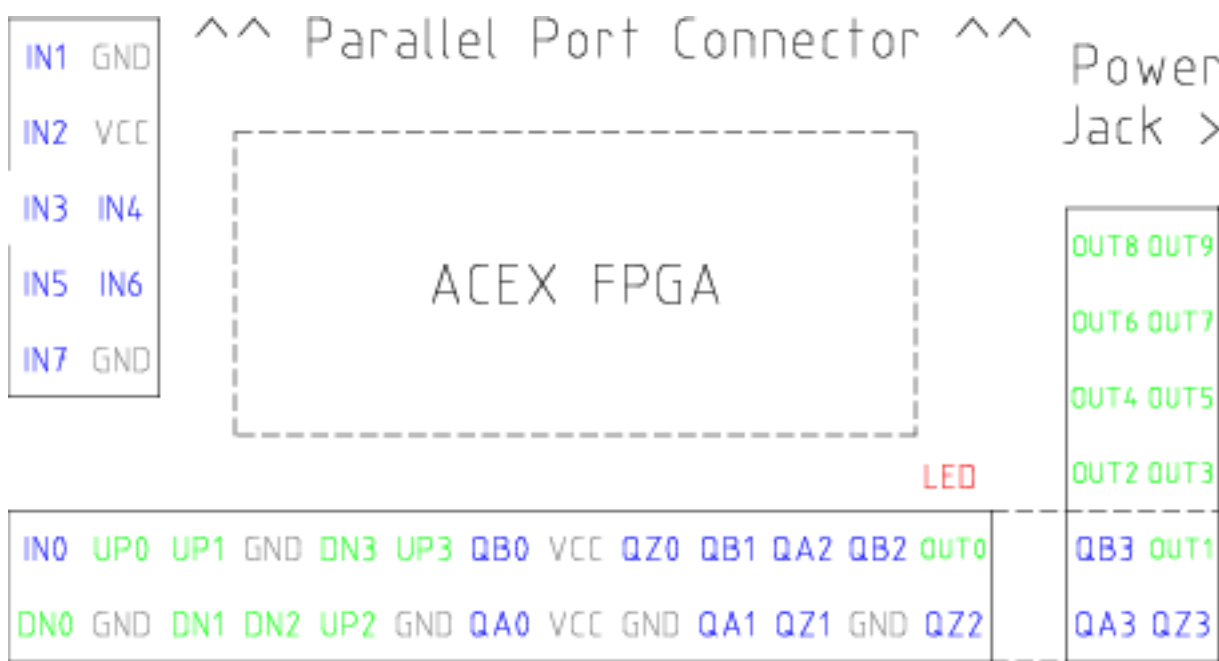


Figure 12.3: Pluto-Servo Pinout

Table 12.2: Pluto-Servo Alternate Pin Functions

Primary function	Alternate Function	Behavior if both functions used
UP0	PWM0	When pwm-0-pwmdir is TRUE, this pin is the PWM output
	OUT10	XOR'd with UP0 or PWM0
UP1	PWM1	When pwm-1-pwmdir is TRUE, this pin is the PWM output
	OUT12	XOR'd with UP1 or PWM1
UP2	PWM2	When pwm-2-pwmdir is TRUE, this pin is the PWM output
	OUT14	XOR'd with UP2 or PWM2
UP3	PWM3	When pwm-3-pwmdir is TRUE, this pin is the PWM output
	OUT16	XOR'd with UP3 or PWM3
DN0	DIR0	When pwm-0-pwmdir is TRUE, this pin is the DIR output
	OUT11	XOR'd with DN0 or DIR0
DN1	DIR1	When pwm-1-pwmdir is TRUE, this pin is the DIR output
	OUT13	XOR'd with DN1 or DIR1
DN2	DIR2	When pwm-2-pwmdir is TRUE, this pin is the DIR output
	OUT15	XOR'd with DN2 or DIR2

Table 12.2: (continued)

Primary function	Alternate Function	Behavior if both functions used
DN3	DIR3	When pwm-3-pwmdir is TRUE, this pin is the DIR output
	OUT17	XOR'd with DN3 or DIR3
QZ0	IN8	Read same value
QZ1	IN9	Read same value
QZ2	IN10	Read same value
QZ3	IN11	Read same value
QA0	IN12	Read same value
QA1	IN13	Read same value
QA2	IN14	Read same value
QA3	IN15	Read same value
QB0	IN16	Read same value
QB1	IN17	Read same value
QB2	IN18	Read same value
QB3	IN19	Read same value

12.9.2.2 Input latching and output updating

- PWM duty cycles for each channel are updated at different times.
- Digital outputs OUT0 through OUT9 are all updated at the same time. Digital outputs OUT10 through OUT17 are updated at the same time as the pwm function they are shared with.
- Digital inputs IN0 through IN19 are all latched at the same time.
- Quadrature positions for each channel are latched at different times.

12.9.2.3 HAL Functions, Pins and Parameters

A list of all *loadrt* arguments, HAL function names, pin names and parameter names is in the manual page, *pluto_servo.9*.

12.9.2.4 Compatible driver hardware

A schematic for a 2A, 2-axis PWM servo amplifier board is available from the ([the software developer](#)). The L298 H-Bridge can be used for motors up to 4A (one motor per L298) or up to 2A (two motors per L298) with the supply voltage up to 46V. However, the L298 does not have built-in current limiting, a problem for motors with high stall currents. For higher currents and voltages, some users have reported success with International Rectifier's integrated high-side/low-side drivers.

12.9.3 Pluto Step

Pluto-step is suitable for control of a 3- or 4-axis CNC mill with stepper motors. The large number of inputs allows for a full set of limit switches.

The board features:

- 4 *step+direction* channels with 312.5 kHz maximum step rate, programmable step length, space, and direction change times
- 14 dedicated digital outputs
- 16 dedicated digital inputs
- EPP communication with the PC

12.9.3.1 Pinout

- *STEP_x* - The *step* (clock) output of stepgen channel *x*
- *DIR_x* - The *direction* output of stepgen channel *x*
- *IN_x* - Dedicated digital input #*x*
- *OUT_x* - Dedicated digital output #*x*
- *GND* - Ground
- *VCC* - +3.3V regulated DC

While the *extended main connector* has a superset of signals usually found on a Step & Direction DB25 connector—4 step generators, 9 inputs, and 6 general-purpose outputs—the layout on this header is different than the layout of a standard 26-pin ribbon cable to DB25 connector.

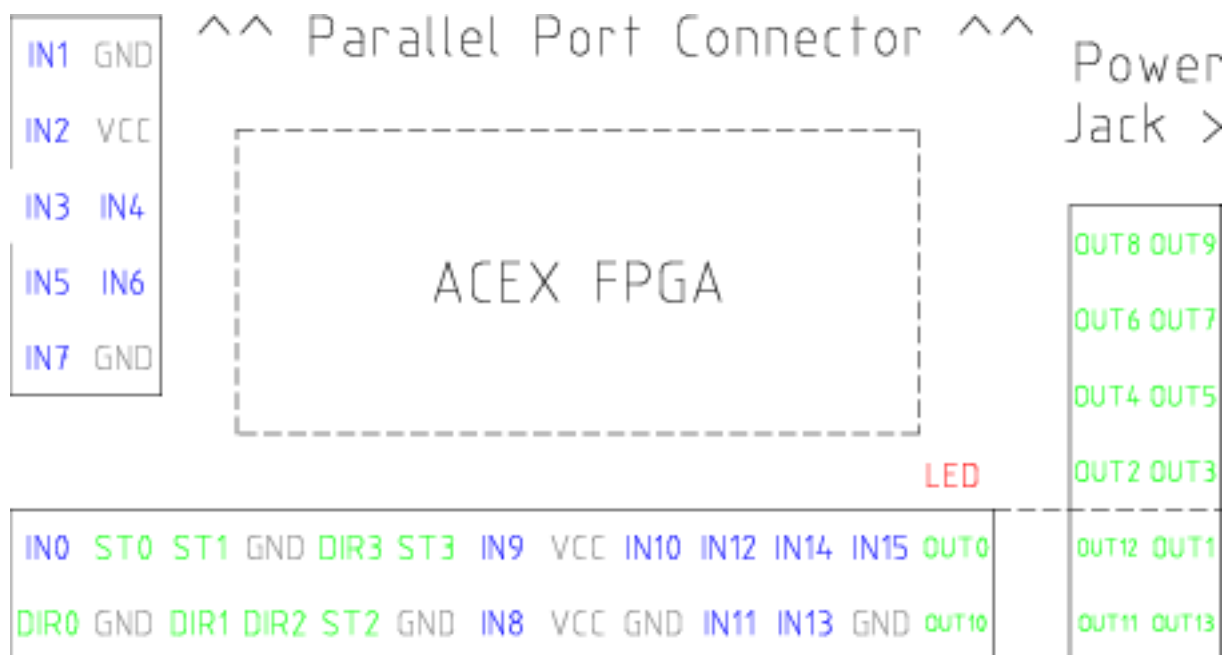


Figure 12.4: Pluto-Step Pinout

12.9.3.2 Input latching and output updating

- Step frequencies for each channel are updated at different times.
- Digital outputs are all updated at the same time.
- Digital inputs are all latched at the same time.
- Feedback positions for each channel are latched at different times.

12.9.3.3 Step Waveform Timings

The firmware and driver enforce step length, space, and direction change times. Timings are rounded up to the next multiple of $1.6\mu\text{s}$, with a maximum of $49.6\mu\text{s}$. The timings are the same as for the software stepgen component, except that *dirhold* and *dirsetup* have been merged into a single parameter *dirtime* which should be the maximum of the two, and that the same step timings are always applied to all channels.

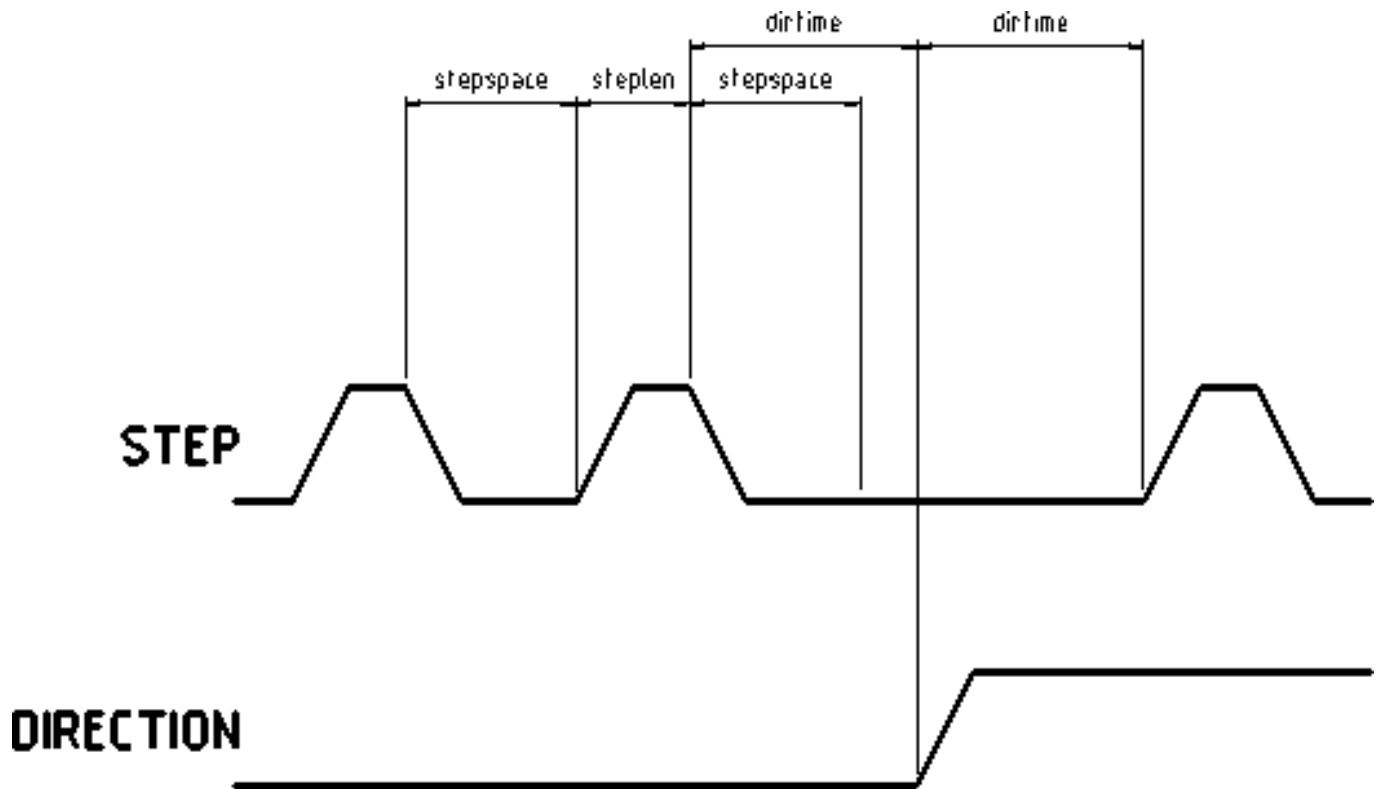


Figure 12.5: Pluto-Step Timings

12.9.3.4 HAL Functions, Pins and Parameters

A list of all *loadrt* arguments, HAL function names, pin names and parameter names is in the manual page, *pluto_step.9*.

12.10 Powermax Modbus Driver

This is a userspace HAL program, written in python, to control Hypetherm Powermax plasma cutters using the Modbus ASCII protocol over RS485.

Note

Since this is a userspace program it can be affected by computer loading and latency. It is possible to lose communications which will be indicated by a change in the status output. One should always have an Estop circuit that kills the power to the unit in case of emergency.

This component is loaded using the halcmd "loadusr" command:

```
loadusr -Wn pmx485 pmx485 /dev/ttyUSB0
```

This will load the pmx485 component using the /dev/ttyUSB0 port and wait for it to become ready. It is necessary to name the port to use for communications.

12.10.1 Pins

- **pmx485.mode-set** (bit, in) # set cutting mode
- **pmx485.current-set** (bit, in) # set cutting current
- **pmx485.pressure-set** (bit, in) # set gas pressure
- **pmx485.enable** (bit, in) # enable the component
- **pmx485.mode** (bit, out) # cut mode feedback
- **pmx485.current** (bit, out) # cutting current feedback
- **pmx485.pressure** (bit, out) # gas pressure feedback
- **pmx485.fault** (bit, out) # powermax fault code
- **pmx485.status** (bit, out) # connection status
- **pmx485.current-min** (bit, out) # minimum allowed current
- **pmx485.current-max** (bit, out) # maximum allowed current
- **pmx485.pressure-min** (bit, out) # minimum allowed gas pressure
- **pmx485.pressure-max** (bit, out) # maximum allowed gas pressure

12.10.2 Description

To communicate with a Powermax, the component must first be enabled via the **enable** pin and it may then initiate a request to the Powermax by writing a valid string to the following pins:

- **mode-set**
- **current-set**
- **pressure-set**

Note

A **pressure-set** value of zero is valid, the Powermax will then calculate the required pressure internally.

Communications may be validated from the Powermax display or the **status** pin. While in remote mode, the mode, current and pressure may be changed as needed.

To terminate the communications, do one of the following:

- Set all set pins to zero: **mode-set**, **current-set**, and **pressure-set**.
 - Disconnect the Powermax power supply from its power source for approximately 30 seconds. When you power the system back ON, it will no longer be in remote mode.
-

12.10.3 Reference:

- Hypertherm Application Note #807220
"Powermax45 XP/65/85/105/125® Serial Communication Protocol"

12.11 Servo To Go Driver

The Servo-To-Go (STG) is one of the first PC motion control cards supported by LinuxCNC. It is an ISA card and it exists in different flavors (all supported by this driver). The board includes up to 8 channels of quadrature encoder input, 8 channels of analog input and output, 32 bits digital I/O, an interval timer with interrupt and a watchdog. For more information see the [Servo To Go](#) web page.

Note

We have had reports that the opamps on the Servo To Go card do not work with newer ATX power supplies that use modern switch mode DC-DC converters. The failure mode is that STG card outputs a constant voltage regardless of what the driver is commanding it to do. Older ATX power supplies with linear voltage regulators do not have this problem, and work fine with the STG cards.

12.11.1 Installing

```
loadrt hal_stg [base=<address>] [num_chan=<nr>] [dio="<dio-string>"] \  
             [model=<model>]
```

The base address field is optional; if it's not provided the driver attempts to autodetect the board. The num_chan field is used to specify the number of channels available on the card, if not used the 8 axis version is assumed. The digital inputs/outputs configuration is determined by a config string passed to insmod when loading the module. The format consists of a four character string that sets the direction of each group of pins. Each character of the direction string is either "I" or "O". The first character sets the direction of port A (Port A - DIO.0-7), the next sets port B (Port B - DIO.8-15), the next sets port C (Port C - DIO.16-23), and the fourth sets port D (Port D - DIO.24-31). The model field can be used in case the driver doesn't autodetect the right card version.

HINT: after starting up the driver, *dmesg* can be consulted for messages relevant to the driver (e.g. autodetected version number and base address). For example:

```
loadrt hal_stg base=0x300 num_chan=4 dio="IOIO"
```

This example installs the STG driver for a card found at the base address of 0x300, 4 channels of encoder feedback, DACs and ADCs, along with 32 bits of I/O configured like this: the first 8 (Port A) configured as Input, the next 8 (Port B) configured as Output, the next 8 (Port C) configured as Input, and the last 8 (Port D) configured as Output

```
loadrt hal_stg
```

This example installs the driver and attempts to autodetect the board address and board model, it installs 8 axes by default along with a standard I/O setup: Port A & B configured as Input, Port C & D configured as Output.

12.11.2 Pins

- *stg.<channel>.counts* - (s32) Tracks the counted encoder ticks.
-

- *stg.<channel>.position* - (float) Outputs a converted position.
- *stg.<channel>.dac-value* - (float) Drives the voltage for the corresponding DAC.
- *stg.<channel>.adc-value* - (float) Tracks the measured voltage from the corresponding ADC.
- *stg.in-<pinnum>* - (bit) Tracks a physical input pin.
- *stg.in-<pinnum>-not* - (bit) Tracks a physical input pin, but inverted.
- *stg.out-<pinnum>* - (bit) Drives a physical output pin

For each pin, *<channel>* is the axis number, and *<pinnum>* is the logic pin number of the STG if II00 is defined, there are 16 input pins (in-00 .. in-15) and 16 output pins (out-00 .. out-15), and they correspond to PORTs ABCD (in-00 is PORTA.0, out-15 is PORTD.7).

The in-<pinnum> HAL pin is TRUE if the physical pin is high, and FALSE if the physical pin is low. The in-<pinnum>-not HAL pin is inverted — it is FALSE if the physical pin is high. By connecting a signal to one or the other, the user can determine the state of the input.

12.11.3 Parameters

- *stg.<channel>.position-scale* - (float) The number of counts / user unit (to convert from counts to units).
- *stg.<channel>.dac-offset* - (float) Sets the offset for the corresponding DAC.
- *stg.<channel>.dac-gain* - (float) Sets the gain of the corresponding DAC.
- *stg.<channel>.adc-offset* - (float) Sets the offset of the corresponding ADC.
- *stg.<channel>.adc-gain* - (float) Sets the gain of the corresponding ADC.
- *stg.out-<pinnum>-invert* - (bit) Inverts an output pin.

The -invert parameter determines whether an output pin is active high or active low. If -invert is FALSE, setting the HAL out- pin TRUE drives the physical pin high, and FALSE drives it low. If -invert is TRUE, then setting the HAL out- pin TRUE will drive the physical pin low.

12.11.4 Functions

- *stg.capture-position* - Reads the encoder counters from the axis *<channel>*.
- *stg.write-dacs* - Writes the voltages to the DACs.
- *stg.read-adcs* - Reads the voltages from the ADCs.
- *stg.di-read* - Reads physical in- pins of all ports and updates all HAL in-<pinnum> and in-<pinnum>-not pins.
- *stg.do-write* - Reads all HAL out-<pinnum> pins and updates all physical output pins.

12.12 Shuttle

12.12.1 Description

Shuttle is a non-realtime HAL component that interfaces Contour Design's ShuttleXpress, ShuttlePRO, and ShuttlePRO2 devices with LinuxCNC's HAL.

If the driver is started without command-line arguments, it will probe all /dev/hidraw* device files for Shuttle devices, and use all devices found. If it is started with command-line arguments, it will only probe the devices specified.

The ShuttleXpress has five momentary buttons, a 10 counts/revolution jog wheel with detents, and a 15-position spring-loaded outer wheel that returns to center when released.

The ShuttlePRO has 13 momentary buttons, a 10 counts/revolution jog wheel with detents, and a 15-position spring-loaded outer wheel that returns to center when released.

The ShuttlePRO2 has 15 momentary buttons, a 10 counts/revolution jog wheel with detents, and a 15-position spring-loaded outer wheel that returns to center when released.

Warning



The Shuttle devices have an internal 8-bit counter for the current jog-wheel position. The shuttle driver can not know this value until the Shuttles device sends its first event. When the first event comes into the driver, the driver uses the device's reported jog-wheel position to initialize counts to 0.

This means that if the first event is generated by a jog-wheel move, that first move will be lost. Any user interaction with the Shuttle device will generate an event, informing the driver of the jog-wheel position. So if you (for example) push one of the buttons at startup, the jog-wheel will work fine and notice the first click.

12.12.2 Setup

The shuttle driver needs read permission to the /dev/hidraw* device files. This can be accomplished by adding a file /etc/udev/rules.d/99-shuttle.rules, with the following contents:

```
SUBSYSTEM=="hidraw", ATTRS{idVendor}=="0b33", ATTRS{idProduct}=="0020", MODE="0444"
SUBSYSTEM=="hidraw", ATTRS{idVendor}=="05f3", ATTRS{idProduct}=="0240", MODE="0444"
SUBSYSTEM=="hidraw", ATTRS{idVendor}=="0b33", ATTRS{idProduct}=="0030", MODE="0444"
```

The LinuxCNC Debian package installs an appropriate udev file automatically, but if you are building LinuxCNC from source and are not using the Debian packaging you'll need to install this file by hand. If you install the file by hand you'll need to tell udev to reload its rules files by running `udevadm control --reload-rules`.

12.12.3 Pins

All HAL pin names are prefixed with `shuttle` followed by the index of the device (the order in which the driver found them), for example "shuttle.0" or "shuttle.2".

<Prefix>.button-<ButtonNumber> (bit out)

These pins are True (1) when the button is pressed.

<Prefix>.button-<ButtonNumber>-not (bit out)

These pins have the inverse of the button state, so they're True (1) when the button is not pressed.

<Prefix>.counts (s32 out)

Accumulated counts from the jog wheel (the inner wheel).

<Prefix>.spring-wheel-s32 (s32 out)

The current deflection of the spring-wheel (the outer wheel). It's 0 at rest, and ranges from -7 at the counter-clockwise extreme to +7 at the clockwise extreme.

<Prefix>.spring-wheel-f (float out)

The current deflection of the spring-wheel (the outer wheel). It's 0.0 at rest, -1.0 at the counter-clockwise extreme, and +1.0 at the clockwise extreme. (The Shuttle devices report the spring-wheel position as an integer from -7 to +7, so this pin reports only 15 discrete values in it's range.)

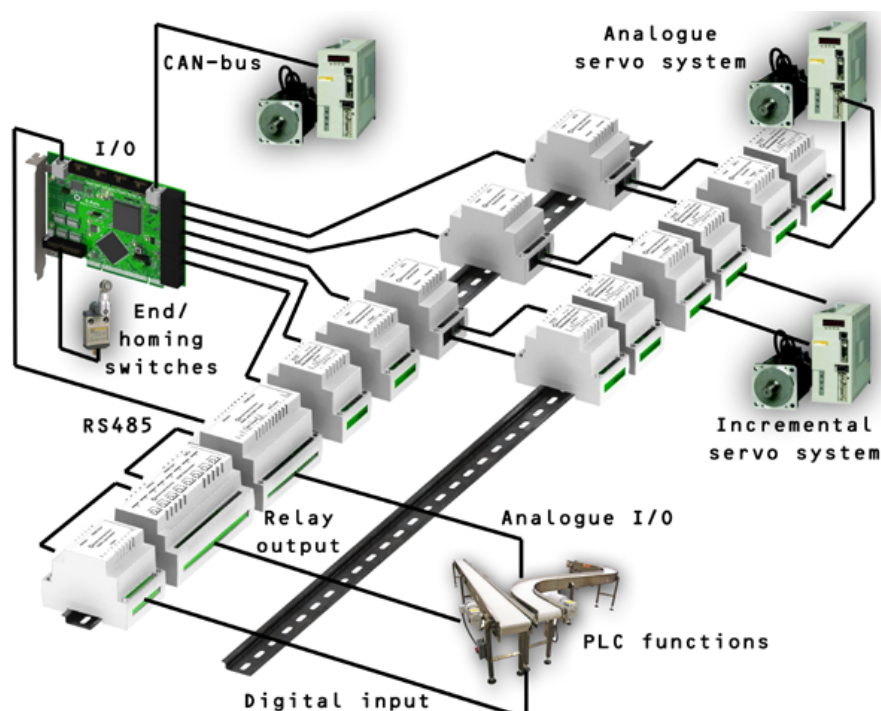
12.13 General Mechatronics Driver

General Mechatronics GM6-PCI card based motion control system

For detailed description, please refer to the [System integration manual](#).

The GM6-PCI motion control card is based on an FPGA and a PCI bridge interface ASIC. A small automated manufacturing cell can be controlled, with a short time system integration procedure. The following figure demonstrating the typical connection of devices related to the control system:

- It can control up to six axis, each can be stepper or CAN bus interface or analogue servo.
- GPIO: Four time eight I/O pins are placed on standard flat cable headers.
- RS485 I/O expander modules: RS485 bus was designed for interfacing with compact DIN-rail mounted expander modules. An 8-channel digital input, an 8-channel relay output and an analogue I/O (4x +/-10 Volts output and 8x +/-5 Volts input) modules are available now. Up to 16 modules can be connected to the bus altogether.
- 20 optically isolated input pins: Six times three for the direct connection of two end switch and one homing sensor for each joint. And additionally, two optically isolated E-stop inputs.



Installing:

```
loadrt hal_gm
```

During loading (or attempted loading) the driver prints some useful debugging messages to the kernel log, which can be viewed with `dmesg`.

Up to 3 boards may be used in one system.

The following connectors can be found on the GM6-PCI card:

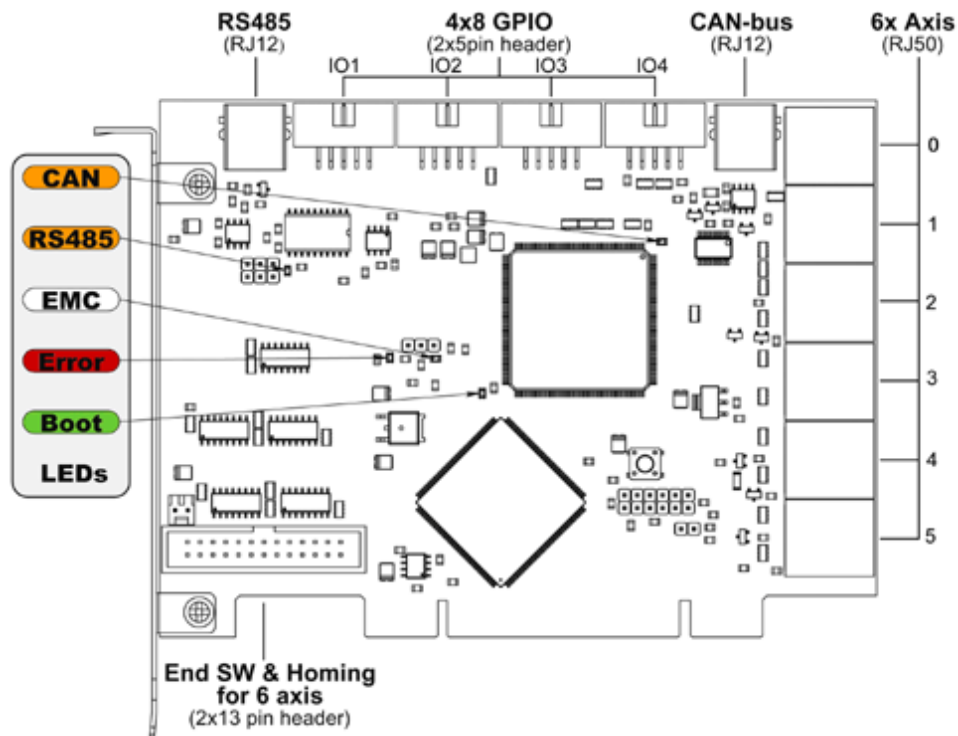


Figure 12.6: GM6-PCI card connectors and LEDs

12.13.1 I/O connectors

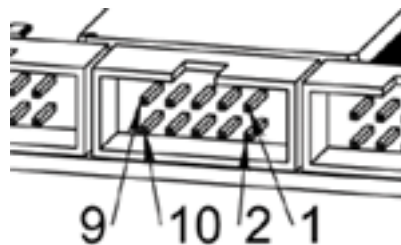


Figure 12.7: Pin numbering of GPIO connectors

Table 12.3: Pinout of GPIO connectors

9	7	5	3	1
IOx/7	IOx/5	IOx/3	IOx/1	VCC

10	8	6	4	2
GND	IOx/6	IOx/4	IOx/2	IOx/0

Each pin can be configured as digital input or output. GM6-PCI motion control card has 4 general purpose I/O (GPIO) connectors, with eight configurable I/O on each. Every GPIO pin and parameter name begins as follows:

```
gm.<nr of card>.gpio.<nr of gpio con>
```

where <nr of gpio con> is from 0 to 3. For example:

```
gm.0.gpio.0.in-0
```

indicates the state of the first pin of the first GPIO connector on the GM6-PCI card. HAL pins are updated by function

```
gm.<nr of card>.read
```

12.13.1.1 Pins

Table 12.4: GPIO pins

Pins	Type and direction	Pin description
.in-<0-7>	(bit, Out)	Input pin
.in-not-<0-7>	(bit, Out)	Negated input pin
.out-<0-7>	(bit, In)	Output pin. Used only when GPIO is set to output.

12.13.1.2 Parameters

Table 12.5: GPIO parameters

Pins	Type and direction	Parameter description
.is-out-<0-7>	(bit, R/W)	When True, the corresponding GPIO is set to totem-pole output, other wise set to high impedance input.
.invert-out-<0-7>	(bit, R/W)	When True, pin value will be inverted. Used when pin is configured as output.

12.13.2 Axis connectors

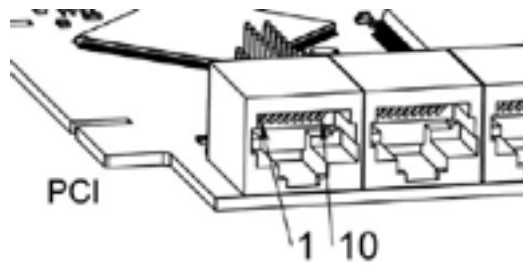


Figure 12.8: Pin numbering of axis connectors

Table 12.6: Pinout of axis connectors

1	Encoder A
2	+5 Volt (PC)
3	Encoder B
4	Encoder Index
5	Fault
6	Power Enabled
7	Step/CCW/B
8	Direction/CW/A
9	Ground (PC)
10	DAC serial line

12.13.2.1 Axis interface modules

Small sized DIN rail mounted interface modules gives easy way of connecting different types of servo modules to the axis connectors. Seven different system configurations are presented in the [System integration manual](#) for evaluating typical applications. Also the detailed description of the Axis modules can be found in the System integration manual.

For evaluating the appropriate servo-drive structure the modules have to be connected as the following block diagram shows:

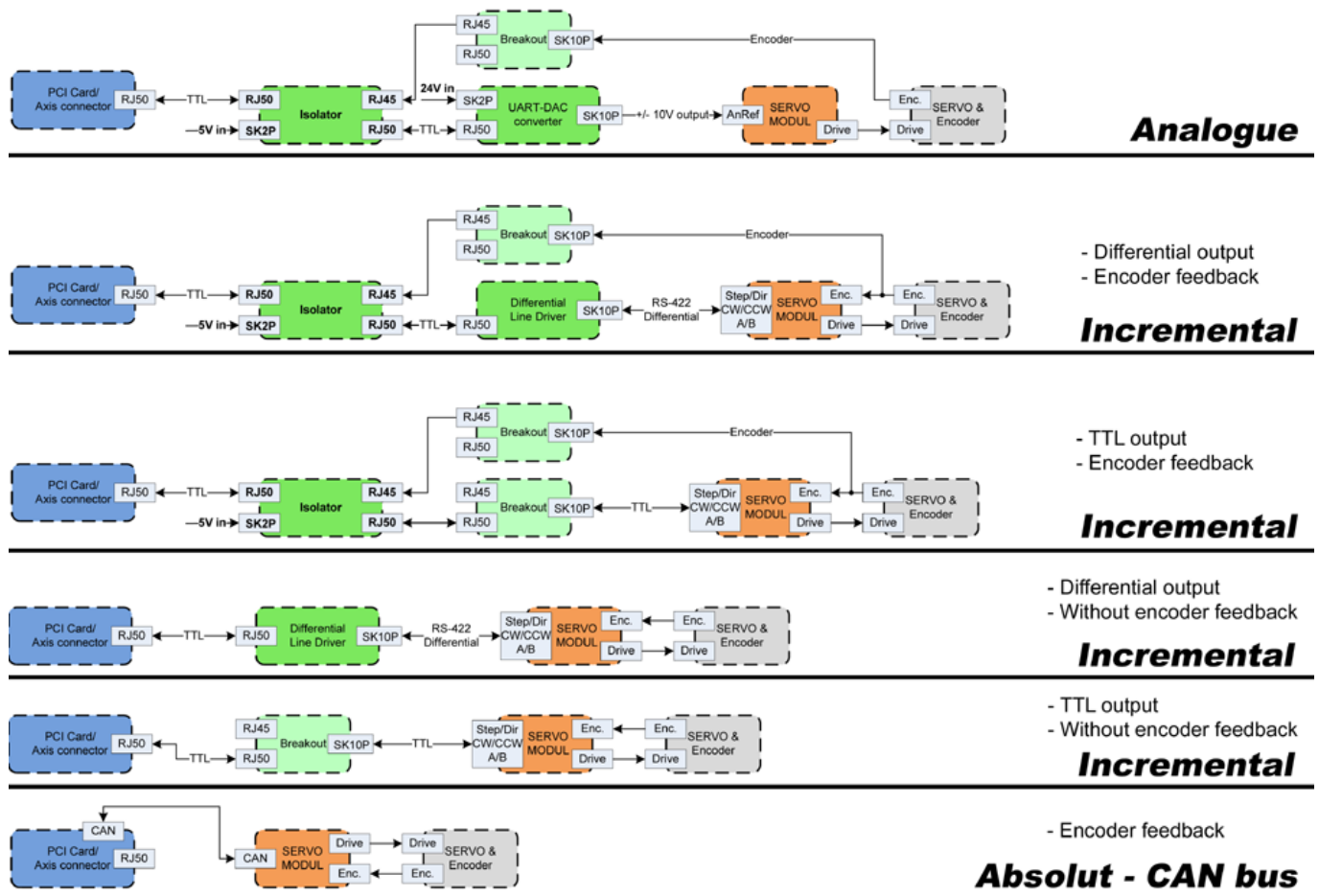


Figure 12.9: Servo axis interfaces

12.13.2.2 Encoder

The GM6-PCI motion control card has six encoder modules. Each encoder module has three channels:

- Channel-A
- Channel-B
- Channel-I (index)

It is able to count quadrature encoder signals or step/dir signals. Each encoder module is connected to the inputs of the corresponding RJ50 axis connector.

Every encoder pin and parameter name begins as follows:

```
gm.<nr of card>.encoder.<nr of axis>
```

where <nr of axis> is from 0 to 5. For example:

```
gm.0.encoder.0.position
```

refers to the position of encoder module of axis 0.

The GM6-PCI card counts the encoder signal independently from LinuxCNC. HAL pins are updated by function:

gm.<nr of card>.read

Table 12.7: Encoder pins

Pins	Type and direction	Pin description
.reset	(bit, In)	When True, resets counts and position to zero.
.rawcounts	(s32, Out)	The raw count is the counts, but unaffected by reset or the index pulse.
.counts	(s32, Out)	Position in encoder counts.
.position	(float, Out)	Position in scaled units (= .counts/.position-scale).
.index-enabled	(bit, IO)	When True, counts and position are rounded or reset (depends on index-mode) on next rising edge of channel-I. Every time position is reset because of Index, index-enabled pin is set to 0 and remain 0 until connected HAL pin does not set it.
.velocity	(float, Out)	Velocity in scaled units per second. GM encoder uses high frequency hardware timer to measure time between encoder pulses in order to calculate velocity. It greatly reduces quantization noise as compared to simply differentiating the position output. When the measured velocity is below min-speed-estimate, the velocity output is 0.

Table 12.8: Encoder parameters

Parameters	Type and Read/Write	Parameter description
.counter-mode	(bit, R/W)	When True, the counter counts each rising edge of the channel-A input to the direction determined by channel-B. This is useful for counting the output of a single channel (non-quadrature) or step/dir signal sensor. When false, it counts in quadrature mode.

Table 12.8: (continued)

Parameters	Type and Read/Write	Parameter description
.index-mode	(bit, R/W)	When True and .index-enabled is also true, .counts and .position are rounded (based on .counts-per-rev) at rising edge of channel-I. This is useful to correct few pulses error caused by noise. In round mode, it is essential to set .counts-per-rev parameter correctly. When .index-mode is False and .index-enabled is true, .counts and .position are reset at channel-I pulse.
.counts-per-rev	(s32, R/V)	Determine how many counts are between two index pulses. It is used only in round mode, so when both .index-enabled and .index-mode parameters are True. GM encoder process encoder signal in 4x mode, so for example in case of a 500 CPR encoder it should be set to 2000. This parameter can be easily measured by setting .index-enabled True and .index-mode False (so that .counts resets at channel-I pulse), than move axis by hand and see the maximum magnitude of .counts pin in halmeter.
.index-invert	(bit, R/W)	When True, channel-I event (reset or round) occur on falling edge of channel-I signal, otherwise on rising edge.
.min-speed-estimate	(float, R/W)	Determine the minimum measured velocity magnitude at which .velocity will be set as nonzero. Setting this parameter too low will cause it to take a long time for velocity to go to zero after encoder pulses have stopped arriving.
.position-scale	(float, R/W)	Scale in counts per length unit. .position=.counts/.position-scale. For example, if position-scale is 2000, then 1000 counts of the encoder will produce a position of 0.5 units.

Connect encoder position to LinuxCNC position feedback:

```
net Xpos-fb gm.0.encoder.0.position => joint.0.motor-pos-fb
```

12.13.2.3 StepGen module

The GM6-PCI motion control card has six StepGen modules, one for each joint. Each module has two output signals. It can produce Step/Direction, Up/Down or Quadrature (A/B) pulses. Each StepGen module is connected to the pins of the corresponding RJ50 axis connector.

Every StepGen pin and parameter name begins as follows:

```
gm.<nr of card>.stepgen.<nr of axis>
```

where nr of axis is from 0 to 5. For example:

```
gm.0.stepgen.0.position-cmd
```

refers to the position command of StepGen module of axis 0 on card 0.

The GM6-PCI card generates step pulses independently from LinuxCNC. HAL pins are updated by function

```
gm.<nr of card>.write
```

Table 12.9: Stepgen module pins

Pins	Type and direction	Pin description
.enable	(bit, In)	Stepgen produces pulses only when this pin is true.
.count-fb	(s32, Out)	Position feedback in counts unit.
.position-fb	(float, Out)	Position feedback in position unit.
.position-cmd	(float, In)	Commanded position in position units. Used in position mode only.
.velocity-cmd	(float, In)	Commanded velocity in position units per second. Used in velocity mode only.

Table 12.10: Stepgen module parameters

Parameters	Type and Read/Write	Parameter description
.step-type	(u32, R/W)	When 0, module produces Step/Dir signal. When 1, it produces Up/Down step signals. And when it is 2, it produces quadrature output signals.
.control-type	(bit, R/W)	When True, .velocity-cmd is used as reference and velocity control calculate pulse rate output. When False, .position-cmd is used as reference and position control calculate pulse rate output.

Table 12.10: (continued)

Parameters	Type and Read/Write	Parameter description
.invert-step1	(bit, R/W)	Invert the output of channel 1 (Step signal in StepDir mode)
.invert-step2	(bit, R/W)	Invert the output of channel 2 (Dir signal in StepDir mode)
.maxvel	(float, R/W)	Maximum velocity in position units per second. If it is set to 0.0, .maxvel parameter is ignored.
.maxaccel	(float, R/W)	Maximum acceleration in position units per second squared. If it is set to 0.0, .maxaccel parameter is ignored.
.position-scale	(float, R/W)	Scale in steps per length unit.
.steplen	(u32, R/W)	Length of step pulse in nano-seconds.
.stepspace	(u32, R/W)	Minimum time between two step pulses in nano-seconds.
.dirdelay	(u32, R/W)	Minimum time between step pulse and direction change in nano-seconds.

For evaluating the appropriate values see the timing diagrams below:

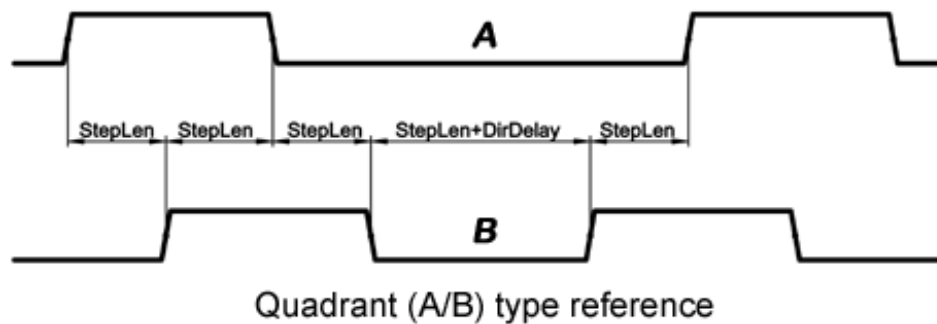
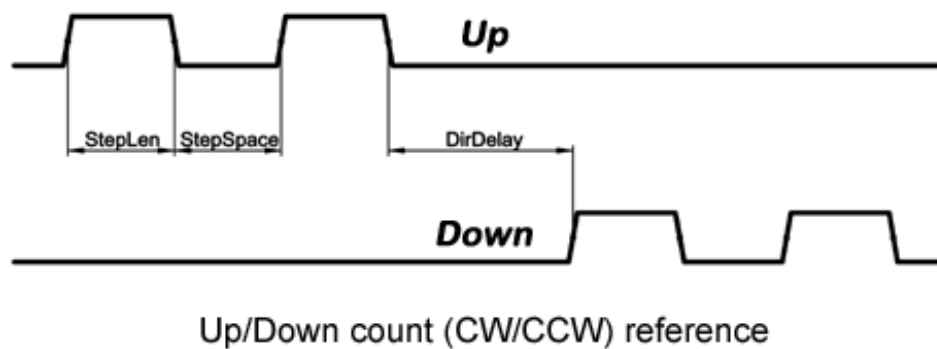
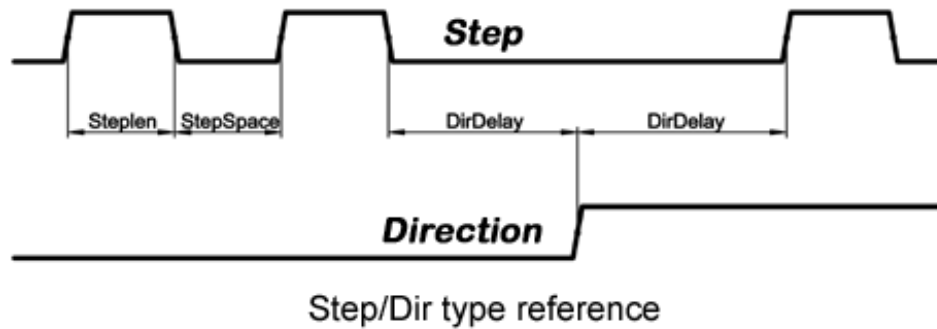


Figure 12.10: Reference signal timing diagrams

HAL example Setting StepGen module of axis 0 to generate 1000 step pulse per position unit:

```
setp gm.0.stepgen.0.step-type 0      # 0:stepDir, 1:UpDown, 2:Quad
setp gm.0.stepgen.0.control-type 0   # 0:Pos. control, 1:Vel. Control
setp gm.0.stepgen.0.invert-step1 0
setp gm.0.stepgen.0.invert-step2 0
setp gm.0.stepgen.0.maxvel 0         # do not set maxvel for step
                                     # generator, let interpolator control it.
setp gm.0.stepgen.0.maxaccel 0       # do not set max acceleration for
                                     # step generator, let interpolator control it.
setp gm.0.stepgen.0.position-scale 1000 # 1000 step/position unit
setp gm.0.stepgen.0.steplen 1000     # 1000 ns = 1 μs
setp gm.0.stepgen.0.stepspace1000   # 1000 ns = 1 μs
```

```
setp gm.0.stepgen.0.dirdelay 2000      # 2000 ns = 2 μs
```

Connect StepGen to axis 0 position reference and enable pins:

```
net Xpos-cmd joint.0.motor-pos-cmd => gm.0.stepgen.0.position-cmd
net Xen joint.0.amp-enable-out => gm.0.stepgen.0.enable
```

12.13.2.4 Enable and Fault signals

The GM6-PCI motion control card has one enable output and one fault input HAL pins, both are connected to each RJ50 axis connector and to the CAN connector.

HAL pins are updated by function:

```
gm.<nr of card>.read
```

Table 12.11: Enable and Fault signal pins

Pins	Type and direction	Pin description
gm.<nr of card>.power-enable	(bit, In)	If this pin is True, * and Watch Dog Timer is not expired * and there is no power fault Then power enable pins of axis- and CAN connectors are set to high, otherwise set to low.
gm.<nr of card>.power-fault	(bit, Out)	Power fault input.

12.13.2.5 Axis DAC

The GM6-PCI motion control card has six serial axis DAC driver modules, one for each joint. Each module is connected to the pin of the corresponding RJ50 axis connector. Every axis DAC pin and parameter name begins as follows:

```
gm.<nr of card>.dac.<nr of axis>
```

where nr of axis is from 0 to 5. For example:

```
gm.0.dac.0.value
```

refers to the output voltage of DAC module of axis 0. HAL pins are updated by function:

```
gm.<nr of card>.write
```

Table 12.12: Axis DAC pins

Pins	Type and direction	Pin description
.enable	(bit, In)	Enable DAC output. When enable is false, DAC output is 0.0 V.
.value	(float, In)	Value of DAC output in Volts.

Table 12.13: Axis DAC parameters

Parameters	Type and direction	Parameter description
.offset	(float, R/W)	Offset is added to the value before the hardware is updated
.high-limit	(float, R/W)	Maximum output voltage of the hardware in volts.
.low-limit	(float, R/W)	Minimum output voltage of the hardware in volts.
.invert-serial	(float, R/W)	GM6-PCI card is communicating with DAC hardware via fast serial communication to highly reduce time delay compared to PWM. DAC module is recommended to be isolated which is negating serial communication line. In case of isolation, leave this parameter to default (0), while in case of none-isolation, set this parameter to 1.

12.13.3 CAN-bus servo amplifiers

The GM6-PCI motion control card has CAN module to drive CAN servo amplifiers. Implementation of higher level protocols like CANopen is further development. Currently GM produced power amplifiers has upper level driver which export pins and parameters to HAL. They receive position reference and provide encoder feedback via CAN bus.

The frames are standard (11 bit) ID frames, with 4 byte data length. The baud rate is 1 Mbit. The position command IDs for axis 0..5 are 0x10..0x15. The position feedback IDs for axis 0..5 are 0x20..0x25.

These configuration can be changed with the modification of `hal_gm.c` and recompiling LinuxCNC.

Every CAN pin and parameter name begins as follows:

```
gm.<nr of card>.can-gm.<nr of axis>
```

where <nr of axis> is from 0 to 5. For example:

```
gm.0.can-gm.0.position
```

refers to the output position of axis 0 in position units.

HAL pins are updated by function:

```
gm.<nr of card>.write
```

12.13.3.1 Pins

Table 12.14: CAN module pins

Pins	Type and direction	Pin description
.enable	(bit, In)	Enable sending position references.
.position-cmd	(float, In)	Commanded position in position units.
.position-fb	(float, In)	Feed back position in position units.

12.13.3.2 Parameters

Table 12.15: CAN module parameters

Parameters	Type and direction	Parameter description
.position-scale	(float, R/W)	Scale in per length unit.

12.13.4 Watchdog timer

Watchdog timer resets at function:

```
gm.<nr of card>.read
```

12.13.4.1 Pins

Table 12.16: Watchdog pins

Pins	Type and direction	Pin description
gm.<nr of card>.watchdog-expired	(bit, Out)	Indicates that watchdog timer is expired.

Watchdog timer overrun causes the set of power-enable to low in hardware.

12.13.4.2 Parameters

Table 12.17: Watchdog parameters

Parameters	Type and direction	Parameter description
gm.<nr of card>.watchdog-enable	(bit, R/W)	Enable watchdog timer. It is strongly recommended to enable watchdog timer, because it can disables all the servo amplifiers by pulling down all enable signal in case of PC error.
gm.<nr of card>.watchdog-timeout-ns	(float, R/W)	Time interval in within the gm.<nr of card>.read function must be executed. The gm.<nr of card>.read is typically added to servo-thread, so watch timeout is typically set to 3 times of the servo period.

12.13.5 End-, homing- and E-stop switches

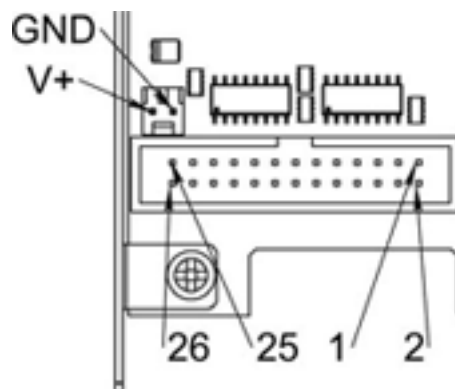


Figure 12.11: Pin numbering of homing & end switch connector

Table 12.18: End- and homing switch connector pinout

25	23	21	19	17	15	13	11	9	7	5	3	1
GND		1/End-	2/End+	2/Hom- ing	3/End-	4/End+	4/Hom- ing	5/End-	6/End+	6/Hom- ing	E- Stop 2	V+ (Ext.)

26	24	22	20	18	16	14	12	10	8	6	4	2
GND		1/End+	1/Hom-ing	2/End-	3/End+	3/Hom-ing	4/End-	5/End+	5/Hom-ing	6/End-	E-Stop 1	V+ (Ext.)

The GM6-PCI motion control card has two limit- and one homing switch input for each joint. All the names of these pins begin as follows:

```
gm.<nr of card>.joint.<nr of axis>
```

where nr of axis is from 0 to 5. For example:

```
gm.0.joint.0.home-sw-in
```

indicates the state of the axis 0 home switch.

HAL pins are updated by function:

```
gm.<nr of card>.read
```

12.13.5.1 Pins

Table 12.19: End- and homing switch pins

Pins	Type and direction	Pin description
.home-sw-in	(bit, Out)	Home switch input
.home-sw-in-not	(bit, Out)	Negated home switch input
.neg-lim-sw-in	(bit, Out)	Negative limit switch input
.neg-lim-sw-in-not	(bit, Out)	Negated negative limit switch input
.pos-lim-sw-in	(bit, Out)	Positive limit switch input
.pos-lim-sw-in-not	(bit, Out)	Negated positive limit switch input

12.13.5.2 Parameters

Table 12.20: E-stop switch parameters

Parameters	Type and direction	Parameter description
gm.0.estop.0.in	(bit, Out)	Estop 0 input
gm.0.estop.0.in-not	(bit, Out)	Negated Estop 0 input
gm.0.estop.1.in	(bit, Out)	Estop 1 input
gm.0.estop.1.in-not	(bit, Out)	Negated Estop 1 input

12.13.6 Status LEDs

12.13.6.1 CAN

Color: Orange

- Blink, during data communication.
- On, when any of the buffers are full - communication error.
- Off, when no data communication.

12.13.6.2 RS485

Color: Orange

- Blink, during initialization of modules on the bus
- On, when the data communication is up between all initialized modules.
- Off, when any of the initialized modules dropped off because of an error.

12.13.6.3 EMC

Color: White

- Blink, when LinuxCNC is running.
- Otherwise off.

12.13.6.4 Boot

Color: Green

- On, when system booted successfully.
- Otherwise off.

12.13.6.5 Error

Color: Red

- Off, when there is no fault in the system.
- Blink, when PCI communication error.
- On, when watchdog timer overflowed.

12.13.7 RS485 I/O expander modules

These modules were developed for expanding the I/O and function capability along an RS485 line of the GM6-PCI motion control card.

Available module types:

- 8-channel relay output module - gives eight NO-NC relay output on a three pole terminal connector for each channel.
 - 8-channel digital input module - gives eight optical isolated digital input pins.
 - 8 channel ADC and 4-channel DAC module - gives four digital-to-analogue converter outputs and eight analogue-to-digital inputs. This module is also optically isolated from the GM6-PCI card.
-

Automatic node recognizing:

Each node connected to the bus was recognized by the GM6-PCI card automatically. During starting LinuxCNC, the driver export pins and parameters of all available modules automatically.

Fault handling:

If a module does not answer regularly the GM6-PCI card drops down the module. If a module with output do not gets data with correct CRC regularly, the module switch to error state (green LED blinking), and turns all outputs to error state.

Connecting the nodes:

The modules on the bus have to be connected in serial topology, with termination resistors on the end. The start of the topology is the PCI card, and the end is the last module.

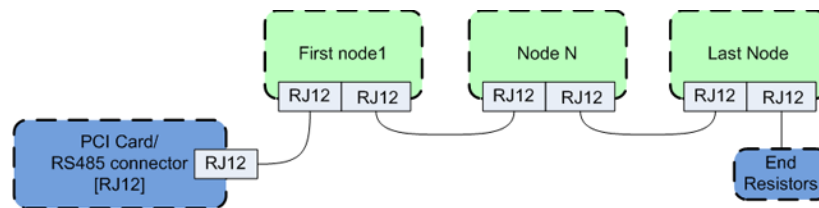


Figure 12.12: Connecting the RS485 nodes to the GM6-PCI card

Addressing:

Each node on the bus has a 4 bit unique address that can be set with a red DIP switch.

Status LED:

A green LED indicates the status of the module:

- Blink, when the module is only powered, but not yet identified, or when module is dropped down.
- Off, during identification (computer is on, but LinuxCNC not started)
- On, when it communicates continuously.

12.13.7.1 Relay output module

For pinout, connection and electrical characteristics of the module, please refer to the [System integration manual](#).

All the pins and parameters are updated by the following function:

```
gm.<nr of card>.rs485
```

It should be added to servo thread or other thread with larger period to avoid CPU overload. Every RS485 module pin and parameter name begins as follows:

```
gm.<nr of card>.rs485.<module ID>
```

where <module ID> is from 00 to 15.

Table 12.21: Relay output module pins

Pins	Type and direction	Pin description
.relay-<0-7>	(bit, Out)	Output pin for relay

Table 12.22: Relay output module parameters

Parameters	Type and direction	Parameter description
.invert-relay-<0-7>	(bit, R/W)	Negate relay output pin

HAL example

```
gm.0.rs485.0.relay-0 # First relay of the node.
gm.0                # Means the first GM6-PCI motion control card (PCI card address = 0)
.rs485.0            # Select node with address 0 on the RS485 bus
.relay-0            # Select the first relay
```

12.13.7.2 Digital input module

For pinout, connection and electrical characteristics of the module, please refer to the [System integration manual](#).

All the pins and parameters are updated by the following function:

```
gm.<nr of card>.rs485
```

It should be added to servo thread or other thread with larger period to avoid CPU overload. Every RS485 module pin and parameter name begins as follows:

```
gm.<nr of card>.rs485.<module ID>
```

where <module ID> is from 00 to 15.

Table 12.23: Digital input output module pins

Pins	Type and direction	Pin description
.in-<0-7>	(bit, Out)	Input pin
.in-not-<0-7>	(bit, Out)	Negated input pin

HAL example

```
gm.0.rs485.0.in-0 # First input of the node.
# gm.0           - Means the first GM6-PCI motion control card (PCI card address = 0)
# .rs485.0       - Select node with address 0 on the RS485 bus
# .in-0          - Select the first digital input module
```

12.13.7.3 DAC & ADC module

For pinout, connection and electrical characteristics of the module, please refer to the [System integration manual](#).

All the pins and parameters are updated by the following function:

```
gm.<nr of card>.rs485
```

It should be added to servo thread or other thread with larger period to avoid CPU overload. Every RS485 module pin and parameter name begins as follows:

```
gm.<nr of card>.rs485.<module ID>
```

where <module ID> is from 00 to 15.

Table 12.24: DAC & ADC module pins

Pins	Type and direction	Pin description
.adc-<0-7>	(float, Out)	Value of ADC input in Volts.
.dac-enable-<0-3>	(bit, In)	Enable DAC output. When enable is false DAC output is set to 0.0 V.
.dac-<0-3>	(float, In)	Value of DAC output in Volts.

Table 12.25: DAC & ADC module parameters

Parameters	Type and direction	Parameter description
.adc-scale-<0-7>	(float, R/W)	The input voltage will be multiplied by scale before being output to .adc- pin.
.adc-offset-<0-7>	(float, R/W)	Offset is subtracted from the hardware input voltage after the scale multiplier has been applied.
.dac-offset-<0-3>	(float, R/W)	Offset is added to the value before the hardware is updated.
.dac-high-limit-<0-3>	(float, R/W)	Maximum output voltage of the hardware in volts.
.dac-low-limit-<0-3>	(float, R/W)	Minimum output voltage of the hardware in volts.

HAL example

```
gm.0.rs485.0.adc-0 # First analogue channel of the node.
# gm.0           - Means the first GM6-PCI motion control card (PCI card address = 0)
# .rs485.0       - Select node with address 0 on the RS485 bus
# .adc-0         - Select the first analogue input of the module
```

12.13.7.4 Teach Pendant module

For pinout, connection and electrical characteristics of the module, please refer to the [System integration manual](#).

All the pins and parameters are updated by the following function:

```
gm.<nr of card>.rs485
```

It should be added to servo thread or other thread with larger period to avoid CPU overload. Every RS485 module pin and parameter name begins as follows:

```
gm.<nr of card>.rs485.<module ID>
```

where <module ID> is from 00 to 15. Note that on the Teach Pendant module it cannot be changed, and pre-programmed as zero. Upon request it can be delivered with firmware pre-programmed different ID.

Table 12.26: Teach Pendant module pins

Pins	Type and direction	Pin description
.adc-<0-5>	(float, Out)	Value of ADC input in Volts.
.enc-reset	(bit, In)	When True, resets counts and position to zero.
.enc-counts	(s32, Out)	Position in encoder counts.
.enc-rawcounts	(s32, Out)	The raw count is the counts, but unaffected by reset.
.enc-position	(float, Out)	Position in scaled units (= .enc-counts/.enc-position-scale).
.in-<0-7>	(bit, Out)	Input pin
.in-not-<0-7>	(bit, Out)	Negated input pin

Table 12.27: Teach Pendant module parameters

Parameters	Type and direction	Parameter description
.adc-scale-<0-5>	(float, R/W)	The input voltage will be multiplied by scale before being output to .adc- pin.
.adc-offset-<0-5>	(float, R/W)	Offset is subtracted from the hardware input voltage after the scale multiplier has been applied.
.enc-position-scale	(float, R/W)	Scale in per length unit.

HAL example

```
gm.0.rs485.0.adc-0 # First analogue channel of the node.
# gm.0           - Means the first GM6-PCI motion control card (PCI card address = 0)
# .rs485.0       - Select node with address 0 on the RS485 bus
# .adc-0         - Select the first analogue input of the module
```

12.13.8 Errata

12.13.8.1 GM6-PCI card Errata

The revision number in this section refers to the revision of the GM6-PCI card device.

Rev. 1.2

- Error: The PCI card do not boot, when Axis 1. END B switch is active (low). Found on November 16, 2013.
- Reason: This switch is connected to a boot setting pin of FPGA
- Problem fix/workaround: Use other switch pin, or connect only normally open switch to this switch input pin.

12.14 VFS11 VFD Driver

This is a userspace HAL program to control the S11 series of VFD's from Toshiba.

vfs11_vfd supports serial and TCP connections. Serial connections may be RS232 or RS485. RS485 is supported in full- and half-duplex mode. TCP connections may be passive (wait for incoming connection), or active outgoing connections, which may be useful to connect to TCP-based devices or through a terminal server.

Regardless of the connection type, vfs11_vfd operates as a Modbus master.

This component is loaded using the halcmd "loadusr" command:

```
loadusr -Wn spindle-vfd vfs11_vfd -n spindle-vfd
```

The above command says: loadusr, wait for named to load, component vfs11_vfd, named spindle-vfd

12.14.1 Command Line Options

vfs11_vfd is mostly configured through INI file options. The command line options are:

- *-n or --name <halname>* : set the HAL component name
- *-I or --ini <inifilename>* : take configuration from this INI file. Defaults to environment variable INI_FILE_NAME.
- *-S or --section <section name>* : take configuration from this section in the INI file. Defaults to VFS11.
- *-d or --debug* enable debug messages on console output.
- *-m or --modbus-debug* enable modbus messages on console output
- *-r or --report-device* report device properties on console at startup

Debugging can be toggled by sending aUSR1 signal to the vfs11_vfd process. Modbus debugging can be toggled by sending aUSR2 signal to vfs11_vfd process (example: kill -USR1 `pidof vfs11_vfd`).

Note

That if there are serial configuration errors, turning on verbose may result in a flood of timeout errors.

12.14.2 Pins

Where `<n>` is `vfs11_vfd` or the name given during loading with the `-n` option.

- `<n>.acceleration-pattern` (bit, in) when true, set acceleration and deceleration times as defined in registers F500 and F501 respectively. Used in PID loops to choose shorter ramp times to avoid oscillation.
- `<n>.alarm-code` (s32, out) non-zero if drive is in alarmed state. Bitmap describing alarm information (see register FC91 description). Use `err-reset` (see below) to clear the alarm.
- `<n>.at-speed` (bit, out) when drive is at commanded speed (see speed-tolerance below)
- `<n>.current-load-percentage` (float, out) reported from the VFD
- `<n>.dc-brake` (bit, in) engage the DC brake. Also turns off spindle-on.
- `<n>.enable` (bit, in) enable the VFD. If false, all operating parameters are still read but control is released and panel control is enabled (subject to VFD setup).
- `<n>.err-reset` (bit, in) reset errors (alarms a.k.a Trip and e-stop status). Resetting the VFD may cause a 2-second delay until it's rebooted and Modbus is up again.
- `<n>.estop` (bit, in) put the VFD into emergency-stopped status. No operation possible until cleared with `err-reset` or powercycling.
- `<n>.frequency-command` (float, out) current target frequency in Hz as set through speed-command (which is in RPM), from the VFD
- `<n>.frequency-out` (float, out) current output frequency of the VFD
- `<n>.inverter-load-percentage` (float, out) current load report from VFD
- `<n>.is-e-stopped` (bit, out) the VFD is in emergency stop status (blinking "E" on panel). Use `err-reset` to reboot the VFD and clear the e- stop status.
- `<n>.is-stopped` (bit, out) true when the VFD reports 0 Hz output
- `<n>.max-rpm` (float, R) actual RPM limit based on maximum frequency the VFD may generate, and the motors nameplate values. For instance, if nameplate-HZ is 50, and nameplate-RPM_ is 1410, but the VFD may generate up to 80 Hz, then max- rpm would read as 2256 (80*1410/50). The frequency limit is read from the VFD at startup. To increase the upper frequency limit, the UL and FH parameters must be changed on the panel. See the VF-S11 manual for instructions how to set the maximum frequency.
- `<n>.modbus-ok` (bit, out) true when the Modbus session is successfully established and the last 10 transactions returned without error.
- `<n>.motor-RPM` (float, out) estimated current RPM value, from the VFD
- `<n>.output-current-percentage` (float, out) from the VFD
- `<n>.output-voltage-percentage` (float, out) from the VFD
- `<n>.output-voltage` (float, out) from the VFD
- `<n>.speed-command` (float, in) speed sent to VFD in RPM. It is an error to send a speed faster than the Motor Max RPM as set in the VFD
- `<n>.spindle-fwd` (bit, in) 1 for FWD and 0 for REV, sent to VFD
- `<n>.spindle-on` (bit, in) 1 for ON and 0 for OFF sent to VFD, only on when running
- `<n>.spindle-rev` (bit, in) 1 for ON and 0 for OFF, only on when running

- `<n>.jog-mode` (bit, in) 1 for ON and 0 for OFF, enables the VF-S11 *jog mode*. Speed control is disabled, and the output frequency is determined by register F262 (preset to 5 Hz). This might be useful for spindle orientation. In normal mode, the VFD shuts off if the frequency drops below 12 Hz.
- `<n>.status` (s32, out) Drive Status of the VFD (see the TOSVERT VF-S11 Communications Function Instruction Manual, register FD01). A bitmap.
- `<n>.trip-code` (s32, out) trip code if VF-S11 is in tripped state.
- `<n>.error-count` (s32, out) number of Modbus transactions which returned an error
- `<n>.max-speed` (bit, in) ignore the loop-time parameter and run Modbus at maximum speed, at the expense of higher CPU usage. Suggested use during spindle positioning.

12.14.3 Parameters

Where `<n>` is `vfs11_vfd` or the name given during loading with the `-n` option.

- `<n>.frequency-limit` (float, RO) upper limit read from VFD setup.
- `<n>.loop-time` (float, RW) how often the Modbus is polled (default interval 0.1 seconds)
- `<n>.nameplate-HZ` (float, RW) Nameplate Hz of motor (default 50). Used to calculate target frequency (together with nameplate-RPM) for a target RPM value as given by speed-command.
- `<n>.nameplate-RPM` (float, RW) Nameplate RPM of motor (default 1410)
- `<n>.rpm-limit` (float, RW) do-not-exceed soft limit for motor RPM (defaults to nameplate-RPM).
- `<n>.tolerance` (float, RW) speed tolerance (default 0.01) for determining whether spindle is at speed (0.01 meaning: output frequency is within 1% of target frequency)

12.14.4 INI file configuration

This lists all options understood by `vfs11_vfd`. Typical setups for RS-232, RS-485 and TCP can be found in `src/hal/user_comps/vfs11_vfd/*.ini`.

```
[VFS11]
# serial connection
TYPE=rtu

# serial port
DEVICE=/dev/ttyS0

# TCP server - wait for incoming connection
TYPE=tcpserver

# tcp portnumber for TYPE=tcpserver or tcpclient
PORT=1502

# TCP client - active outgoing connection
TYPE=tcpclient

# destination to connect to if TYPE=tcpclient
TCPDEST=192.168.1.1

#----- meaningful only if TYPE=rtu -----
# serial device detail
# 5 6 7 8
```

```

BITS= 5

# even odd none
PARITY=none

# 110, 300, 600, 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200
BAUD=19200

# 1 2
STOPBITS=1

#rs232 rs485
SERIAL_MODE=rs485

# up down none
# this feature might not work with a stock Ubuntu
# libmodbus5/libmodbus-dev package, and generate a warning
# execution will continue as if RTS_MODE=up were given.
RTS_MODE=up
#-----

# modbus timers in seconds
# inter-character timer
BYTE_TIMEOUT=0.5
# packet timer
RESPONSE_TIMEOUT=0.5

# target modbus ID
TARGET=1

# on I/O failure, try to reconnect after sleeping
# for RECONNECT_DELAY seconds
RECONNECT_DELAY=1

# misc. parameters
DEBUG=10
MODBUS_DEBUG=0
POLLCYCLES=10

```

12.14.5 HAL example

```

#
# example usage of the VF-S11 VFD driver
#
#
loadusr -Wn spindle-vfd vfs11_vfd -n spindle-vfd

# connect the spindle direction pins to the VFD
net vfs11-fwd spindle-vfd.spindle-fwd <= spindle.0.forward
net vfs11-rev spindle-vfd.spindle-rev <= spindle.0.reverse

# connect the spindle on pin to the VF-S11
net vfs11-run spindle-vfd.spindle-on <= spindle.0.on

# connect the VF-S11 at speed to the motion at speed
net vfs11-at-speed spindle.0.at-speed <= spindle-vfd.at-speed

# connect the spindle RPM to the VF-S11
net vfs11-RPM spindle-vfd.speed-command <= spindle.0.speed-out

```

```
# connect the VF-S11 DC brake
# since this draws power during spindle off, the dc-brake pin would
# better be driven by a monoflop which triggers on spindle-on falling edge
#net vfs11-spindle-brake spindle.N.brake => spindle-vfd.dc-brake

# to use the VFS11 jog mode for spindle orient
# see orient.9 and motion.9
net spindle-orient spindle.0.orient spindle-vfd.max-speed spindle-vfd.jog-mode

# take precedence over control panel
setp spindle-vfd.enable 1
```

12.14.6 Panel operation

The `vfs11_vfd` driver takes precedence over panel control while it is enabled (see *enable* pin), effectively disabling the panel. Clearing the *enable* pin re-enables the panel. Pins and parameters can still be set, but will not be written to the VFD until the *enable* pin is set. Operating parameters are still read while bus control is disabled. Exiting the `vfs11_vfd` driver in a controlled way will release the VFD from the bus and restore panel control.

See the LinuxCNC Integrators Manual for more information. For a detailed register description of the Toshiba VFD's, see the "TOSVERT VF-S11 Communications Function Instruction Manual" (Toshiba document number E6581222) and the "TOSVERT VF-S11 Instruction manual" (Toshiba document number E6581158).

12.14.7 Error Recovery

`vfs11_vfd` recovers from I/O errors as follows: First, all HAL pins are set to default values, and the driver will sleep for `RECONNECT_DELAY` seconds (default 1 second).

- Serial (TYPE=rtu) mode: on error, close and reopen the serial port.
- TCP server (TYPE=tcpsrvr) mode: on losing the TCP connection, the driver will go back to listen for incoming connections.
- TCP client (TYPE=tcpcnt) mode: on losing the TCP connection, the driver will reconnect to `TCPDEST:PORTNO`.

12.14.8 Configuring the VFS11 VFD for Modbus usage

12.14.8.1 Connecting the Serial Port

The VF-S11 has an RJ-45 jack for serial communication. Unfortunately, it does not have a standard RS-232 plug and logic levels. The Toshiba-recommended way is: connect the USB001Z USB-to-serial conversion unit to the drive, and plug the USB port into the PC. A cheaper alternative is a homebrew interface ([hints from Toshiba support](#), [circuit diagram](#)).

Note: the 24V output from the VFD has no short-circuit protection.

Serial port factory defaults are 9600/8/1/even, the protocol defaults to the proprietary "Toshiba Inverter Protocol".

12.14.8.2 Modbus setup

Several parameters need setting before the VF-S11 will talk to this module. This can either be done manually with the control panel, or over the serial link - Toshiba supplies a Windows application called *PCM001Z* which can read/set parameters in the VFD. Note - PCM001Z only talks the Toshiba inverter protocol. So the last parameter which you'd want to change is the protocol - set from Toshiba Inverter Protocol to Modbus; thereafter, the Windows app is useless.

To increase the upper frequency limit, the UL and FH parameters must be changed on the panel. I increased them from 50 to 80.

See dump-params.mio for a description of non-standard VF-S11 parameters of my setup. This file is for the [modio Modbus interactive utility](#).

12.14.9 Programming Note

The vfs11_vfd driver uses the [libmodbus version 3](#) library which is more recent than the version 2 code used in gs2_vfd.

The Ubuntu libmodbus5 and libmodbus-dev packages are only available starting from Ubuntu 12 (*Precise Pengolin*). Moreover, these packages lack support for the MODBUS_RTS_MODE_* flags. Therefore, building vfs11_vfd using this library might generate a warning if RTS_MODE= is specified in the INI file.

To use the full functionality on lucid and precise:

- remove the libmodbus packages: `sudo apt-get remove libmodbus5 libmodbus-dev`
- build and install libmodbus version 3 from source as outlined [here](#).

Libmodbus does not build on Ubuntu Hardy, hence vfs11_vfd is not available on Hardy.

Chapter 13

Driver Examples

13.1 PCI Parallel Port

When you add a second parallel port to your PCI bus you have to find out the address before you can use it with LinuxCNC.

To find the address of your parallel port card open a terminal window and type

```
lspci -v
```

You will see something similar to this as well as info on everything else on the PCI bus:

```
0000:00:10.0 Communication controller: \
    NetMos Technology PCI 1 port parallel adapter (rev 01)
    Subsystem: LSI Logic / Symbios Logic: Unknown device 0010
    Flags: medium devsel, IRQ 11
    I/O ports at a800 [size=8]
    I/O ports at ac00 [size=8]
    I/O ports at b000 [size=8]
    I/O ports at b400 [size=8]
    I/O ports at b800 [size=8]
    I/O ports at bc00 [size=16]
```

In my case the address was the first one so I changed my .hal file from

```
loadrt hal_parport cfg=0x378
```

to

```
loadrt hal_parport cfg="0x378 0xa800 in"
```

(Note the double quotes surrounding the addresses.)

and then added the following lines so the parport will be read and written:

```
addf parport.1.read base-thread
addf parport.1.write base-thread
```

After doing the above then run your config and verify that the parallel port got loaded in Machine/Show HAL Configuration window.

13.2 Spindle Control

LinuxCNC can control up to 8 spindles. The number is set in the INI file. The examples below all refer to a single-spindle config with spindle control pins with names like spindle.0... In the case of a multiple spindle machine all that changes is that additional pins exist with names such as spindle.6...

13.2.1 0-10 Volt Spindle Speed

If your spindle speed is controlled by an analog signal, (for example, by a VFD with a 0 V to 10 V signal) and you're using a DAC card like the m5i20 to output the control signal:

First you need to figure the scale of spindle speed to control signal, i.e. the voltage. For this example the spindle top speed of 5000 RPM is equal to 10 volts.

$$\frac{10 \text{ Volts}}{5000 \text{ RPM}} = \frac{0.002 \text{ Volts}}{1 \text{ RPM}}$$

We have to add a scale component to the HAL file to scale the spindle.N.speed-out to the 0 to 10 needed by the VFD if your DAC card does not do scaling.

```
loadrt scale count=1
addf scale.0 servo-thread
setp scale.0.gain 0.002
net spindle-speed-scale spindle.0.speed-out => scale.0.in
net spindle-speed-DAC scale.0.out => <your DAC pin name>
```

13.2.2 PWM Spindle Speed

If your spindle can be controlled by a PWM signal, use the pwmgen component to create the signal:

```
loadrt pwmgen output_type=0
addf pwmgen.update servo-thread
addf pwmgen.make-pulses base-thread
net spindle-speed-cmd spindle.0.speed-out => pwmgen.0.value
net spindle-on spindle.0.on => pwmgen.0.enable
net spindle-pwm pwmgen.0.pwm => parport.0.pin-09-out
# Set the spindle's top speed in RPM
setp pwmgen.0.scale 1800
```

This assumes that the spindle controller's response to PWM is simple: 0% PWM gives 0 RPM, 10% PWM gives 180 RPM, etc. If there is a minimum PWM required to get the spindle to turn, follow the example in the nist-lathe sample configuration to use a scale component.

13.2.3 Spindle Enable

If you need a spindle enable signal, link your output pin to spindle.0.on. To link these pins to a parallel port pin put something like the following in your .hal file, making sure you pick the pin that is connected to your control device.

```
net spindle-enable spindle.0.on => parport.0.pin-14-out
```

13.2.4 Spindle Direction

If you have direction control of your spindle, then the HAL pins *spindle.N.forward* and *spindle.N.reverse* are controlled by the G-codes M3 and M4. Spindle speed *Sn* must be set to a positive non-zero value for M3/M4 to turn on spindle motion.

To link these pins to a parallel port pin, put something like the following in your .hal file making sure you pick the pin that is connected to your control device.

```
net spindle-fwd spindle.0.forward => parport.0.pin-16-out
net spindle-rev spindle.0.reverse => parport.0.pin-17-out
```

13.2.5 Spindle Soft Start

If you need to ramp your spindle speed command and your control does not have that feature it can be done in HAL. Basically you need to hijack the output of *spindle.N.speed-out* and run it through a *limit2* component with the scale set so it will ramp the rpm from *spindle.N.speed-out* to your device that receives the rpm. The second part is to let LinuxCNC know when the spindle is at speed so motion can begin.

In the 0-10 volt example the line

```
net spindle-speed-scale spindle.0.speed-out => scale.0.in
```

is changed as shown in the following example:

13.2.5.1 Intro to HAL components *limit2* and *near*

In case you have not run across them before, here's a quick introduction to the two HAL components used in the following example.

- A "limit2" is a HAL component (floating point) that accepts an input value and provides an output that has been limited to a max/min range, and also limited to not exceed a specified rate of change.
- A "near" is a HAL component (floating point) with a binary output that says whether two inputs are approximately equal.

More info is available in the documentation for HAL components, or from the man pages, just say *man limit2* or *man near* in a terminal.

```
# load the real time modules limit2 and near with names so it is easier to follow their ↵
connections
loadrt limit2 names=spindle-ramp
loadrt near names=spindle-at-speed

# add the functions to a thread
addf spindle-ramp servo-thread
addf spindle-at-speed servo-thread

# set the parameter for max rate-of-change
# (max spindle accel/decel in units per second)
setp spindle-ramp.maxv 60

# hijack the spindle speed out and send it to spindle ramp in
net spindle-cmd <= spindle.0.speed-out => spindle-ramp.in

# the output of spindle ramp is sent to the scale in
net spindle-ramped <= spindle-ramp.out => scale.0.in
```

```
# to know when to start the motion we send the near component
# (named spindle-at-speed) to the spindle commanded speed from
# the signal spindle-cmd and the actual spindle speed
# provided your spindle can accelerate at the maxv setting.
net spindle-cmd => spindle-at-speed.in1
net spindle-ramped => spindle-at-speed.in2

# the output from spindle-at-speed is sent to spindle.0.at-speed
# and when this is true motion will start
net spindle-ready <= spindle-at-speed.out => spindle.0.at-speed
```

13.2.6 Spindle Feedback

13.2.6.1 Spindle Synchronized Motion

Spindle feedback is needed by LinuxCNC to perform any spindle coordinated motions like threading and constant surface speed. LinuxCNC can perform synchronized motion and CSS with any of up to 8 spindles. Which spindles are used is controlled from G-code. CSS is possible with several spindles simultaneously.

The StepConf Wizard can perform the connections for a single-spindle configuration for you if you select Encoder Phase A and Encoder Index as inputs.

Hardware assumptions for this example:

- An encoder is connected to the spindle and puts out 100 pulses per revolution on phase A.
- The encoder A phase is connected to the parallel port pin 10.
- The encoder index pulse is connected to the parallel port pin 11.

Basic Steps to add the components and configure them: [1](#) [2](#) [3](#)

```
# Add the encoder to HAL and attach it to threads.
loadrt encoder num_chan=4
addf encoder.update-counters base-thread
addf encoder.capture-position servo-thread

# Set the HAL encoder to 100 pulses per revolution.
setp encoder.3.position-scale 100

# Set the HAL encoder to non-quadrature simple counting using A only.
setp encoder.3.counter-mode true

# Connect the HAL encoder outputs to LinuxCNC.
net spindle-position encoder.3.position => spindle.0.revs
net spindle-velocity encoder.3.velocity => spindle.0.speed-in
net spindle-index-enable encoder.3.index-enable <=> spindle.0.index-enable

# Connect the HAL encoder inputs to the real encoder.
net spindle-phase-a encoder.3.phase-A <= parport.0.pin-10-in
net spindle-phase-b encoder.3.phase-B
net spindle-index encoder.3.phase-Z <= parport.0.pin-11-in
```

¹In this example, we will assume that some encoders have already been issued to axes/joints 0, 1, and 2. So the next encoder available for us to attach to the spindle would be number 3. Your situation may differ.

²The HAL encoder index-enable is an exception to the rule in that it behaves as both an input and an output, see the [Encoder Section](#) for details

³It is because we selected *non-quadrature simple counting*... above that we can get away with *quadrature* counting without having any B quadrature input.

13.2.6.2 Spindle At Speed

To enable LinuxCNC to wait for the spindle to be at speed before executing a series of moves, the spindle.N.at-speed needs to turn true at the moment the spindle is at the commanded speed. To achieve this you need spindle feedback from an encoder. Since the feedback and the commanded speed are not usually *exactly* the same you should to use the *near* component to determine that the two numbers are close enough.

The connections needed are from the spindle velocity command signal to near.n.in1 and from the spindle velocity from the encoder to near.n.in2. Then the near.n.out is connected to spindle.N.at-speed. The near.n.scale needs to be set to say how close the two numbers must be before turning on the output. Depending on your setup you may need to adjust the scale to work with your hardware.

The following is typical of the additions needed to your HAL file to enable Spindle At Speed. If you already have near in your HAL file then increase the count and adjust code to suit. Check to make sure the signal names are the same in your HAL file.

```
# load a near component and attach it to a thread
loadrt near
addf near.0 servo-thread

# connect one input to the commanded spindle speed
net spindle-cmd => near.0.in1

# connect one input to the encoder-measured spindle speed
net spindle-velocity => near.0.in2

# connect the output to the spindle-at-speed input
net spindle-at-speed spindle.0.at-speed <= near.0.out

# set the spindle speed inputs to agree if within 1%
setp near.0.scale 1.01
```

13.3 MPG Pendant

This example is to explain how to hook up the common MPG pendants found on the market today. This example uses an MPG3 pendant and a C22 pendant interface card from CNC4PC connected to a second parallel port plugged into the PCI slot. This example gives you 3 axes with 3 step increments of 0.1, 0.01, 0.001

In your custom.hal file or jog.hal file add the following, making sure you don't have mux4 or an encoder already in use. If you do just increase the counts and change the reference numbers. More information about mux4 and encoder can be found in the HAL manual or the man page.

See the [HAL Ini Section](#) of the manual for more information on adding a hal file.

Jog management pins are provided for each joint and all coordinate letters. This example uses the axis jog pins for jogging in world mode. Machines with non-identity kinematics may need use additional connections for jogging in joint mode.

jog.hal

```
# Jog Pendant
loadrt encoder num_chan=1
loadrt mux4 count=1
addf encoder.capture-position servo-thread
addf encoder.update-counters base-thread
addf mux4.0 servo-thread

# If your MPG outputs a quadrature signal per click set x4 to 1
```

```

# If your MPG puts out 1 pulse per click set x4 to 0
setp encoder.0.x4-mode 0

# For velocity mode, set to 1
# In velocity mode the axis stops when the dial is stopped
# even if that means the commanded motion is not completed,
# For position mode (the default), set to 0
# In position mode the axis will move exactly jog-scale
# units for each count, regardless of how long that might take,
setp axis.x.jog-vel-mode 0
setp axis.y.jog-vel-mode 0
setp axis.z.jog-vel-mode 0

# This sets the scale that will be used based on the input to the mux4
setp mux4.0.in0 0.1
setp mux4.0.in1 0.01
setp mux4.0.in2 0.001

# The inputs to the mux4 component
net scale1 mux4.0.sel0 <= parport.1.pin-09-in
net scale2 mux4.0.sel1 <= parport.1.pin-10-in

# The output from the mux4 is sent to each axis jog scale
net mpg-scale <= mux4.0.out
net mpg-scale => axis.x.jog-scale
net mpg-scale => axis.y.jog-scale
net mpg-scale => axis.z.jog-scale

# The MPG inputs
net mpg-a encoder.0.phase-A <= parport.1.pin-02-in
net mpg-b encoder.0.phase-B <= parport.1.pin-03-in

# The Axis select inputs
net mpg-x axis.x.jog-enable <= parport.1.pin-04-in
net mpg-y axis.y.jog-enable <= parport.1.pin-05-in
net mpg-z axis.z.jog-enable <= parport.1.pin-06-in

# The encoder output counts to the axis. Only the selected axis will move.
net encoder-counts <= encoder.0.counts
net encoder-counts => axis.x.jog-counts
net encoder-counts => axis.y.jog-counts
net encoder-counts => axis.z.jog-counts

```

If the machine is capable of high acceleration to smooth out the moves for each click of the MPG use the [ilowpass](#) component to limit the acceleration.

jog.hal with ilowpass

```

loadrt encoder num_chan=1
loadrt mux4 count=1
addf encoder.capture-position servo-thread
addf encoder.update-counters base-thread
addf mux4.0 servo-thread

loadrt ilowpass
addf ilowpass.0 servo-thread

setp ilowpass.0.scale 1000
setp ilowpass.0.gain 0.01

# If your MPG outputs a quadrature signal per click set x4 to 1
# If your MPG puts out 1 pulse per click set x4 to 0

```

```

setp encoder.0.x4-mode 0

# For velocity mode, set to 1
# In velocity mode the axis stops when the dial is stopped
# even if that means the commanded motion is not completed,
# For position mode (the default), set to 0
# In position mode the axis will move exactly jog-scale
# units for each count, regardless of how long that might take,
setp axis.x.jog-vel-mode 0
setp axis.y.jog-vel-mode 0
setp axis.z.jog-vel-mode 0

# This sets the scale that will be used based on the input to the mux4
# The scale used here has to be multiplied by the ilowpass scale
setp mux4.0.in0 0.0001
setp mux4.0.in1 0.00001
setp mux4.0.in2 0.000001

# The inputs to the mux4 component
net scale1 mux4.0.sel0 <= parport.1.pin-09-in
net scale2 mux4.0.sel1 <= parport.1.pin-10-in

# The output from encoder counts is sent to ilowpass
net mpg-out ilowpass.0.in <= encoder.0.counts

# The output from the mux4 is sent to each axis jog scale
net mpg-scale <= mux4.0.out
net mpg-scale => axis.x.jog-scale
net mpg-scale => axis.y.jog-scale
net mpg-scale => axis.z.jog-scale

# The MPG inputs
net mpg-a encoder.0.phase-A <= parport.1.pin-02-in
net mpg-b encoder.0.phase-B <= parport.1.pin-03-in

# The Axis select inputs
net mpg-x axis.x.jog-enable <= parport.1.pin-04-in
net mpg-y axis.y.jog-enable <= parport.1.pin-05-in
net mpg-z axis.z.jog-enable <= parport.1.pin-06-in

# The output from the ilowpass is sent to each axis jog count
# Only the selected axis will move.
net encoder-counts <= ilowpass.0.out
net encoder-counts => axis.x.jog-counts
net encoder-counts => axis.y.jog-counts
net encoder-counts => axis.z.jog-counts

```

13.4 GS2 Spindle

13.4.1 Example

This example shows the connections needed to use an Automation Direct GS2 VFD to drive a spindle. The spindle speed and direction is controlled by LinuxCNC.

Using the GS2 component involves very little to set up. We start with a StepConf Wizard generated config. Make sure the pins with "Spindle CW" and "Spindle PWM" are set to unused in the parallel port setup screen.

In the custom.hal file we place the following to connect LinuxCNC to the GS2 and have LinuxCNC control the drive.

GS2 Example

```
# load the user space component for the Automation Direct GS2 VFD's
loadusr -Wn spindle-vfd gs2_vfd -r 9600 -p none -s 2 -n spindle-vfd

# connect the spindle direction pin to the GS2
net gs2-fwd spindle-vfd.spindle-fwd <= spindle.N.forward

# connect the spindle on pin to the GS2
net gs2-run spindle-vfd.spindle-on <= spindle.N.on

# connect the GS2 at speed to the motion at speed
net gs2-at-speed spindle.N.at-speed <= spindle-vfd.at-speed

# connect the spindle RPM to the GS2
net gs2-RPM spindle-vfd.speed-command <= spindle.N.speed-out
```

Note

The transmission speed might be able to be faster depending on the exact environment. Both the drive and the command line options must match. To check for transmission errors add the -v command line option and run from a terminal.

On the GS2 drive itself you need to set a couple of things before the modbus communications will work. Other parameters might need to be set based on your physical requirements but these are beyond the scope of this manual. Refer to the GS2 manual that came with the drive for more information on the drive parameters.

- The communications switches must be set to RS-232C.
- The motor parameters must be set to match the motor.
- P3.00 (Source of Operation Command) must be set to Operation determined by RS-485 interface, 03 or 04.
- P4.00 (Source of Frequency Command) must be set to Frequency determined by RS232C/RS485 communication interface, 05.
- P9.01 (Transmission Speed) must be set to 9600 baud, 01.
- P9.02 (Communication Protocol) must be set to "Modbus RTU mode, 8 data bits, no parity, 2 stop bits", 03.

A PyVCP panel based on this example is [here](#).

Chapter 14

PLC

14.1 ClassicLadder Introduction

14.1.1 History

ClassicLadder is a free implementation of a ladder interpreter, released under the LGPL. It was written by Marc Le Douarain.

He describes the beginning of the project on his website:

I decided to program a ladder language only for test purposes at the start, in February 2001. It was planned, that I would have to participate to a new product after leaving the enterprise in which I was working at that time. And I was thinking that to have a ladder language in those products could be a nice option to considerate. And so I started to code the first lines for calculating a rung with minimal elements and displaying dynamically it under Gtk, to see if my first idea to realize all this works.

And as quickly I've found that it advanced quite well, I've continued with more complex elements: timer, multiples rungs, etc...

Voila, here is this work... and more: I've continued to add features since then.

— Marc Le Douarain, from "Genesis" at the ClassicLadder website

ClassicLadder has been adapted to work with LinuxCNC's HAL, and is currently being distributed along with LinuxCNC. If there are issues/problems/bugs please report them to the LinuxCNC project.

14.1.2 Introduction

Ladder logic or the Ladder programming language is a method of drawing electrical logic schematics. It is now a graphical language very popular for programming Programmable Logic Controllers (PLCs). It was originally invented to describe logic made from relays. The name is based on the observation that programs in this language resemble ladders, with two vertical *rails* and a series of horizontal *rungs* between them. In Germany and elsewhere in Europe, the style is to draw the rails horizontally along the top and bottom of the page while the rungs are drawn vertically from left to right.

A program in ladder logic, also called a ladder diagram, is similar to a schematic for a set of relay circuits. Ladder logic is useful because a wide variety of engineers and technicians can understand and use it without much additional training because of the resemblance.

Ladder logic is widely used to program PLCs, where sequential control of a process or manufacturing operation is required. Ladder logic is useful for simple but critical control systems, or for reworking

old hardwired relay circuits. As programmable logic controllers became more sophisticated it has also been used in very complex automation systems.




Ladder logic can be thought of as a rule-based language, rather than a procedural language. A *rung* in the ladder represents a rule. When implemented with relays and other electromechanical devices, the various rules *execute* simultaneously and immediately. When implemented in a programmable logic controller, the rules are typically executed sequentially by software, in a loop. By executing the loop fast enough, typically many times per second, the effect of simultaneous and immediate execution is obtained.

Ladder logic follows these general steps for operation.

- Read Inputs
- Solve Logic
- Update Outputs

14.1.3 Example

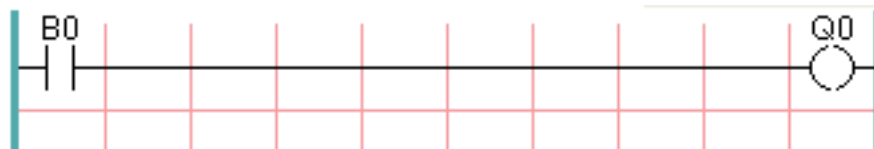
The most common components of ladder are contacts (inputs), these usually are either NC (normally closed) or NO (normally open), and coils (outputs).

- the NO contact 
- the NC contact 
- the coil (output) 

Of course there are many more components to a full ladder language, but understanding these will help you grasp the overall concept.

The ladder consists of one or more rungs. These rungs are horizontal traces (representing wires), with components on them (inputs, outputs and other), which get evaluated left to right.

This example is the simplest rung:



The input on the left, B0, a normally open contact, is connected to the coil (output) on the right, Q0. Now imagine a voltage gets applied to the leftmost end, because the input B0 turns true (e.g. the input is activated, or the user pushed the NO contact). The voltage has a direct path to reach the coil (output) on the right, Q0. As a consequence, the Q0 coil (output) will turn from 0/off/false to 1/on/true. If the user releases B0, the Q0 output quickly returns to 0/off/false.

14.1.4 Basic Latching On-Off Circuit

Building on the above example, suppose we add a switch that closes whenever the coil Q0 is active. This would be the case in a relay, where the coil can activate the switch contacts; or in a contactor, where there are often several small auxiliary contacts in addition to the large 3-phase contacts that are the primary feature of the contactor.

Since this auxiliary switch is driven from coil Q0 in our earlier example, we will give it the same number as the coil that drives it. This is the standard practice followed in all ladder programming, although it may seem strange at first to see a switch labeled the same as a coil. So let's call this auxiliary contact Q0 and connect it across the B0 *pushbutton* contact from our earlier example.

Let's take a look at it:



As before, when the user presses pushbutton B0, coil Q0 comes on. And when coil Q0 comes on, switch Q0 comes on. Now the interesting part happens. When the user releases pushbutton B0, coil Q0 does not stop as it did before. This is because switch Q0 of this circuit is effectively holding the user's pushbutton pressed. So we see that switch Q0 is still holding coil Q0 on after the *start* pushbutton has been released.

This type of contact on a coil or relay, used in this way, is often called a *holding contact*, because it *holds on* the coil that it is associated with. It is also occasionally called a *seal* contact, and when it is active it is said that the circuit is *sealed*.

Unfortunately, our circuit so far has little practical use, because, although we have an *on* or *start* button in the form of pushbutton B0, we have no way to shut this circuit off once it is started. But that's easy to fix. All we need is a way to interrupt the power to coil Q0. So let's add a normally-closed (NC) pushbutton just ahead of coil Q0.

Here's how that would look:



Now we have added *off* or *stop* pushbutton B1. If the user pushes it, contact from the rung to the coil is broken. When coil Q0 loses power, it drops to 0/off/false. When coil Q0 goes off, so does switch Q0, so the *holding contact* is broken, or the circuit is *unsealed*. When the user releases the *stop* pushbutton, contact is restored from the rung to coil Q0, but the rung has gone dead, so the coil doesn't come back on.

This circuit has been used for decades on virtually every machine that has a three-phase motor controlled by a contactor, so it was inevitable that it would be adopted by ladder/PLC programmers. It is also a very safe circuit, in that if *start* and *stop* are both pressed at the same time, the *stop* function always wins.

This is the basic building block of much of ladder programming, so if you are new to it, you would do well to make sure that you understand how this circuit operates.

14.2 ClassicLadder Programming

14.2.1 Ladder Concepts

ClassicLadder is a type of programming language originally implemented on industrial PLCs (it's called Ladder Programming). It is based on the concept of relay contacts and coils, and can be used to construct logic checks and functions in a manner that is familiar to many systems integrators. Ladder consists of rungs that may have branches and resembles an electrical circuit. It is important to know how ladder programs are evaluated when running.

It seems natural that each line would be evaluated left to right, then the next line down, etc., but it doesn't work this way in ladder logic. Ladder logic *scans* the ladder rungs 3 times to change the state of the outputs.

- the inputs are read and updated
- the logic is figured out
- the outputs are set

This can be confusing at first if the output of one line is read by the input of a another rung. There will be one scan before the second input becomes true after the output is set.

Another gotcha with ladder programming is the "Last One Wins" rule. If you have the same output in different locations of your ladder the state of the last one will be what the output is set to.

14.2.2 Languages

The most common language used when working with ClassicLadder is *ladder*. ClassicLadder also supports Sequential Function Chart (Grafcet).

14.2.3 Components

There are two components to ClassicLadder.

- The real time module `classicladder_rt`
- The user space module (including a GUI) `classicladder`

14.2.3.1 Files

Typically ClassicLadder components are placed in the `custom.hal` file if your working from a Stepconf generated configuration. These must not be placed in the `custom_postgui.hal` file or the Ladder Editor menu will be grayed out.

Note

Ladder files (.clp) must not contain any blank spaces in the name.

14.2.3.2 Realtime Module

Loading the ClassicLadder real time module (classicladder_rt) is possible from a HAL file, or directly using a halcmd instruction. The first line loads real time the ClassicLadder module. The second line adds the function classicladder.0.refresh to the servo thread. This line makes ClassicLadder update at the servo thread rate.

```
loadrt classicladder_rt
addf classicladder.0.refresh servo-thread
```

The speed of the thread that ClassicLadder is running in directly affects the responsiveness to inputs and outputs. If you can turn a switch on and off faster than ClassicLadder can notice it then you may need to speed up the thread. The fastest that ClassicLadder can update the rungs is one millisecond. You can put it in a faster thread but it will not update any faster. If you put it in a slower than one millisecond thread then ClassicLadder will update the rungs slower. The current scan time will be displayed on the section display, it is rounded to microseconds. If the scan time is longer than one millisecond you may want to shorten the ladder or put it in a slower thread.

14.2.3.3 Variables

It is possible to configure the number of each type of ladder object while loading the ClassicLadder real time module. If you do not configure the number of ladder objects ClassicLadder will use the default values.

Table 14.1: Default Variable Count

Object Name	Variable Name	Default Value
Number of rungs	(numRungs)	100
Number of bits	(numBits)	20
Number of word variables	(numWords)	20
Number of timers	(numTimers)	10
Number of timers IEC	(numTimersIec)	10
Number of monostables	(numMonostables)	10
Number of counters	(numCounters)	10
Number of HAL inputs bit pins	(numPhysInputs)	15
Number of HAL output bit pins	(numPhysOutputs)	15
Number of arithmetic expressions	(numArithmExpr)	50
Number of Sections	(numSections)	10
Number of Symbols	(numSymbols)	Auto
Number of S32 inputs	(numS32in)	10
Number of S32 outputs	(numS32out)	10
Number of Float inputs	(numFloatIn)	10
Number of Float outputs	(numFloatOut)	10

Objects of most interest are numPhysInputs, numPhysOutputs, numS32in, and numS32out.

Changing these numbers will change the number of HAL bit pins available. numPhysInputs and numPhysOutputs control how many HAL bit (on/off) pins are available. numS32in and numS32out control how many HAL signed integers (+- integer range) pins are available.

For example (you don't need all of these to change just a few):

```
loadrt classicladder_rt numRungs=12 numBits=100 numWords=10
numTimers=10 numMonostables=10 numCounters=10 numPhysInputs=10
numPhysOutputs=10 numArithmExpr=100 numSections=4 numSymbols=200
numS32in=5 numS32out=5
```

To load the default number of objects:

```
loadrt classicladder_rt
```

14.2.4 Loading the ClassicLadder user module

ClassicLadder HAL commands must be executed before the GUI loads or the menu item Ladder Editor will not function. If you used the Stepper Config Wizard place any ClassicLadder HAL commands in the custom.hal file.

To load the user module:

```
loadusr classicladder
```

Note

Only one .clp file can be loaded. If you need to divide your ladder then use sections.

To load a ladder file:

```
loadusr classicladder myladder.clp
```

ClassicLadder Loading Options

- *--nogui* - (loads without the ladder editor) normally used after debugging is finished.
- *--modbus_port=port* - (loads the modbus port number)
- *--modmaster* - (initializes MODBUS master) should load the ladder program at the same time or the TCP is default port.
- *--modslave* - (initializes MODBUS slave) only TCP

To use ClassicLadder with HAL without EMC:

```
loadusr -w classicladder
```

The -w tells HAL not to close down the HAL environment until ClassicLadder is finished.

If you first load ladder program with the *--nogui* option then load ClassicLadder again with no options the GUI will display the last loaded ladder program.

In AXIS you can load the GUI from File/Ladder Editor...

14.2.5 ClassicLadder GUI

If you load ClassicLadder with the GUI it will display two windows: section display, and section manager.

14.2.5.1 Sections Manager

When you first start up ClassicLadder you get an empty Sections Manager window.

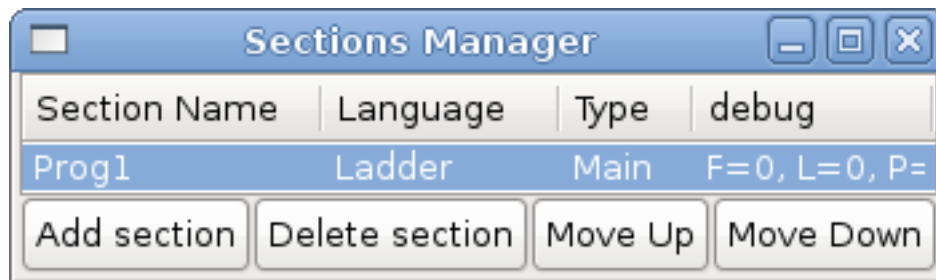


Figure 14.1: Sections Manager Default Window

This window allows you to name, create or delete sections and choose what language that section uses. This is also how you name a subroutine for call coils.

14.2.5.2 Section Display

When you first start up ClassicLadder you get an empty Section Display window. Displayed is one empty rung.



Figure 14.2: Section Display Default Window

Most of the buttons are self explanatory:

The Vars button is for looking at variables, toggle it to display one, the other, both, then none of the windows.

The Config button is used for modbus and shows the max number of ladder elements that was loaded with the real time module.

The Symbols button will display an editable list of symbols for the variables (hint you can name the inputs, outputs, coils etc).

The Quit button will shut down the user program meaning Modbus and the display. The real time ladder program will still run in the background.

The check box at the top right allows you to select whether variable names or symbol names are displayed

You might notice that there is a line under the ladder program display that reads "Project failed to load..." That is the status bar that gives you info about elements of the ladder program that you click on in the display window. This status line will now display HAL signal names for variables %I, %Q and the first %W (in an equation) You might see some funny labels, such as (103) in the rungs. This is displayed (on purpose) because of an old bug- when erasing elements older versions sometimes didn't erase the object with the right code. You might have noticed that the long horizontal connection button sometimes didn't work in the older versions. This was because it looked for the *free* code but found something else. The number in the brackets is the unrecognized code. The ladder program will still work properly, to fix it erase the codes with the editor and save the program.

14.2.5.3 The Variable Windows

These are two variable windows: the Bit Status Window (boolean) and the Watch Window (signed integer). The Vars button is in the Section Display Window, toggle the Vars button to display one, the other, both, then none of the variable windows.

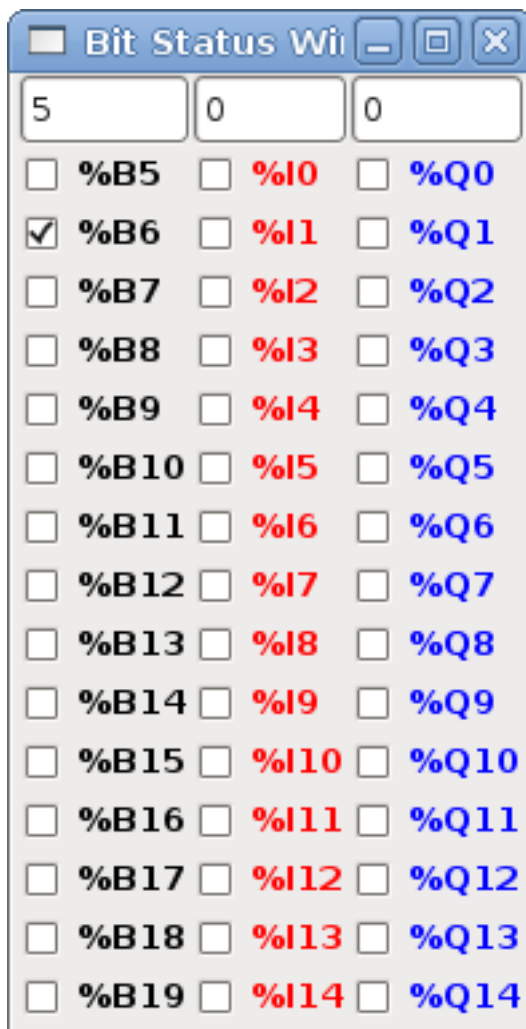


Figure 14.3: Bit Status Window

The Bit Status Window displays some of the boolean (on/off) variable data. Notice all variables start with the % sign. The %I variables represent HAL input bit pins. The %Q represents the relay coil and HAL output bit pins. The %B represents an internal relay coil or internal contact. The three edit areas at the top allow you to select what 15 variables will be displayed in each column. For instance, if the %B Variable column were 15 entries high, and you entered 5 at the top of the column, variables %B5 to %B19 would be displayed. The check boxes allow you to set and unset %B variables manually as long as the ladder program isn't setting them as outputs. Any Bits that are set as outputs by the program when ClassicLadder is running can not be changed and will be displayed as checked if on and unchecked if off.

Watch Window				
Memory	%W0	0	Dec	▼
Bit In Pin	%I1	0	Dec	▼
Bit Out Pin	%Q2	0	Dec	▼
S32in Pin	%IW3	0	Dec	▼
S32out Pin	%QW4	0	Dec	▼
Bit Memory	%B5	0	Dec	▼
IEC Timer	%TM0.Q	0	Dec	▼
IEC Timer	%TM0.V	0	Dec	▼
IEC Timer	%TM0.P	10	Dec	▼
Counter	%C0.D	0	Dec	▼
Counter	%C0.E	0	Dec	▼
Counter	%C0.F	0	Dec	▼
Counter	%C0.V	0	Dec	▼
Counter	%C0.P	0	Dec	▼
Error Bit	%E0	0	Dec	▼

Figure 14.4: Watch Window

The Watch Window displays variable status. The edit box beside it is the number stored in the variable and the drop-down box beside that allow you to choose whether the number to be displayed in hex, decimal or binary. If there are symbol names defined in the symbols window for the word variables showing and the *display symbols* checkbox is checked in the section display window, symbol names will be displayed. To change the variable displayed, type the variable number, e.g. %W2 (if the display symbols check box is not checked) or type the symbol name (if the display symbols checkbox is checked) over an existing variable number/name and press the Enter Key.

14.2.5.4 Symbol Window



Variable	Symbol name	HAL signal/Comment
%I0	%I0	no signal connected
%I1	%I1	no signal connected
%I2	%I2	no signal connected
%I3	%I3	no signal connected
%I4	%I4	no signal connected
%I5	%I5	no signal connected
%I6	%I6	no signal connected
%I7	%I7	no signal connected
%I8	%I8	no signal connected
%I9	%I9	no signal connected

Figure 14.5: Symbol Names window

This is a list of *symbol* names to use instead of variable names to be displayed in the section window when the *display symbols* check box is checked. You add the variable name (remember the % symbol and capital letters), symbol name . If the variable can have a HAL signal connected to it (%I, %Q, and %W-if you have loaded s32 pin with the real time module) then the comment section will show the current HAL signal name or lack thereof. Symbol names should be kept short to display better. Keep in mind that you can display the longer HAL signal names of %I, %Q and %W variable by clicking on them in the section window. Between the two, one should be able to keep track of what the ladder program is connected to!

14.2.5.5 The Editor window



Figure 14.6: Editor Window

- *Add* - adds a rung after the selected rung
- *Insert* - inserts a rung before the selected rung
- *Delete* - deletes the selected rung
- *Modify* - opens the selected rung for editing

Starting from the top left image:

- Object Selector, Eraser
- N.O. Input, N.C. Input, Rising Edge Input , Falling Edge Input

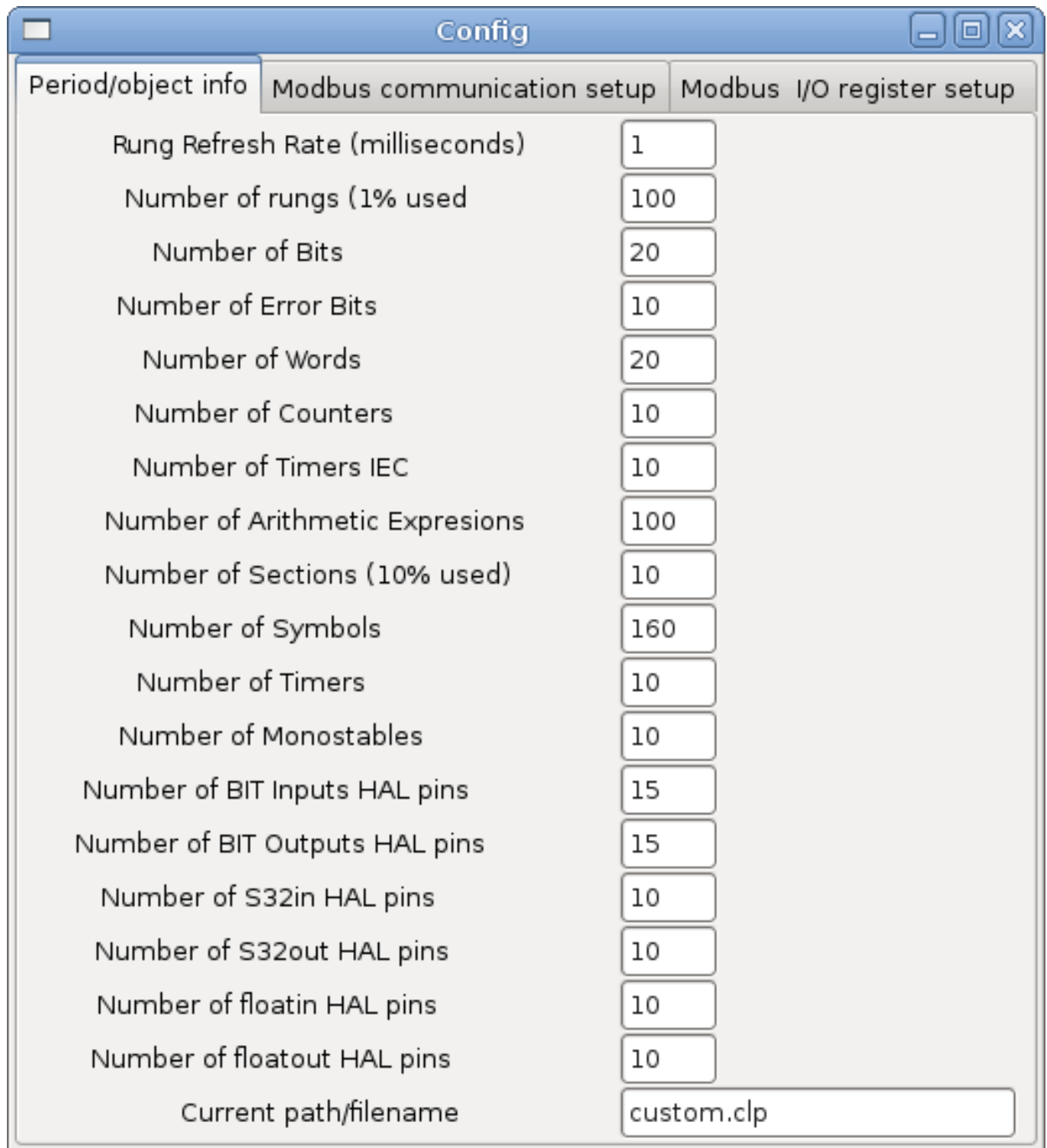
- Horizontal Connection, Vertical Connection , Long Horizontal Connection
- Timer IEC Block, Counter Block, Compare Variable
- Old Timer Block, Old Monostable Block (These have been replaced by the IEC Timer)
- COILS - N.O. Output, N.C. Output, Set Output, Reset Output
- Jump Coil, Call Coil, Variable Assignment

A short description of each of the buttons:

- *Selector* - allows you to select existing objects and modify the information.
- *Eraser* - erases an object.
- *N.O. Contact* - creates a normally open contact. It can be an external HAL-pin (%I) input contact, an internal-bit coil (%B) contact or a external coil (%Q) contact. The HAL-pin input contact is closed when the HAL-pin is true. The coil contacts are closed when the corresponding coil is active (%Q2 contact closes when %Q2 coil is active).
- *N.C. Contact* - creates a normally closed contact. It is the same as the N.O. contact except that the contact is open when the HAL-pin is true or the coil is active.
- *Rising Edge Contact* - creates a contact that is closed when the HAL-pin goes from False to true, or the coil from not-active to active.
- *Falling Edge Contact* - creates a contact that is closed when the HAL-pin goes from true to false or the coil from active to not.
- *Horizontal Connection* - creates a horizontal connection to objects.
- *Vertical Connection* - creates a vertical connection to horizontal lines.
- *Horizontal Running Connection* - creates a horizontal connection between two objects and is a quick way to connect objects that are more than one block apart.
- *IEC Timer* - creates a timer and replaces the *Timer*.
- *Timer* - creates a Timer Module (depreciated use IEC Timer instead).
- *Monostable* - creates a one-shot monostable module
- *Counter* - creates a counter module.
- *Compare* - creates a compare block to compare variable to values or other variables. (eg %W1<=5 or %W1=%W2) Compare cannot be placed in the right most side of the section display.
- *Variable Assignment* - creates an assignment block so you to assign values to variables. (eg %W2=7 or %W1=%W2) ASSIGNMENT functions can only be placed at the right most side of the section display.

14.2.5.6 Config Window

The config window shows the current project status and has the Modbus setup tabs.



Period/object info	Modbus communication setup	Modbus I/O register setup
Rung Refresh Rate (milliseconds)	1	
Number of rungs (1% used)	100	
Number of Bits	20	
Number of Error Bits	10	
Number of Words	20	
Number of Counters	10	
Number of Timers IEC	10	
Number of Arithmetic Expressions	100	
Number of Sections (10% used)	10	
Number of Symbols	160	
Number of Timers	10	
Number of Monostables	10	
Number of BIT Inputs HAL pins	15	
Number of BIT Outputs HAL pins	15	
Number of S32in HAL pins	10	
Number of S32out HAL pins	10	
Number of floatin HAL pins	10	
Number of floatout HAL pins	10	
Current path/filename	custom.clp	



Figure 14.7: Config Window

14.2.6 Ladder objects

14.2.6.1 CONTACTS

Represent switches or relay contacts. They are controlled by the variable letter and number assigned to them.

The variable letter can be B, I, or Q and the number can be up to a three digit number eg. %I2, %Q3, or %B123. Variable I is controlled by a HAL input pin with a corresponding number. Variable B is for internal contacts, controlled by a B coil with a corresponding number. Variable Q is controlled by a Q coil with a corresponding number. (like a relay with multiple contacts). E.g. if HAL pin classicladder.0.in-00 is true then %I0 N.O. contact would be on (closed, true, whatever you like to call it). If %B7 coil is *energized* (on, true, etc) then %B7 N.O. contact would be on. If %Q1 coil is *energized* then %Q1 N.O. contact would be on (and HAL pin classicladder.0.out-01 would be true.)

- *N.O. Contact* -  (Normally Open) When the variable is false the switch is off.
- *N.C. Contact* -  (Normally Closed) When the variable is false the switch is on.
- *Rising Edge Contact* - When the variable changes from false to true, the switch is PULSED on.
- *Falling Edge Contact* - When the variable changes from true to false, the switch is PULSED on.

14.2.6.2 IEC TIMERS

Represent new count down timers. IEC Timers replace Timers and Monostables.

IEC Timers have 2 contacts.

- *I* - input contact
- *Q* - output contact

There are three modes - TON, TOF, TP.

- *TON* - When timer input is true countdown begins and continues as long as input remains true. After countdown is done and as long as timer input is still true the output will be true.
- *TOF* - When timer input is true, sets output true. When the input is false the timer counts down then sets output false.
- *TP* - When timer input is pulsed true or held true timer sets output true till timer counts down. (one-shot)

The time intervals can be set in multiples of 100ms, seconds, or minutes.

There are also Variables for IEC timers that can be read and/or written to in compare or operate blocks.

- %TMxxx.Q - timer done (Boolean, read write)
- %TMxxx.P - timer preset (read write)
- %TMxxx.V - timer value (read write)

14.2.6.3 TIMERS

Represent count down timers. This is deprecated and replaced by IEC Timers.

Timers have 4 contacts.

- *E* - enable (input) starts timer when true, resets when goes false
- *C* - control (input) must be on for the timer to run (usually connect to E)
- *D* - done (output) true when timer times out and as long as E remains true
- *R* - running (output) true when timer is running

The timer base can be multiples of milliseconds, seconds, or minutes.

There are also Variables for timers that can be read and/or written to in compare or operate blocks.

- *%Txx.R* - Timer xx running (Boolean, read only)
- *%Txx.D* - Timer xx done (Boolean, read only)
- *%Txx.V* - Timer xx current value (integer, read only)
- *%Txx.P* - Timer xx preset (integer, read or write)

14.2.6.4 MONOSTABLES

Represent the original one-shot timers. This is now deprecated and replaced by IEC Timers.

Monostables have 2 contacts, I and R.

- *I* - input (input) will start the mono timer running.
- *R* - running (output) will be true while timer is running.

The I contact is rising edge sensitive meaning it starts the timer only when changing from false to true (or off to on). While the timer is running the I contact can change with no effect to the running timer. R will be true and stay true till the timer finishes counting to zero. The timer base can be multiples of milliseconds, seconds, or minutes.

There are also Variables for monostables that can be read and/or written to in compare or operate blocks.

- *%Mxx.R* - Monostable xx running (Boolean, read only)
- *%Mxx.V* - Monostable xx current value (integer, read only)
- *%Mxx.P* - Monostable xx preset (integer, read or write)

14.2.6.5 COUNTERS

Represent up/down counters.

There are 7 contacts:

- *R* - reset (input) will reset the count to 0.
 - *P* - preset (input) will set the count to the preset number assigned from the edit menu.
 - *U* - up count (input) will add one to the count.
-

- *D* - down count (input) will subtract one from the count.
- *E* - under flow (output) will be true when the count rolls over from 0 to 9999.
- *D* - done (output) will be true when the count equals the preset.
- *F* - overflow (output) will be true when the count rolls over from 9999 to 0.

The up and down count contacts are edge sensitive meaning they only count when the contact changes from false to true (or off to on if you prefer).

The range is 0 to 9999.

There are also Variables for counters that can be read and/or written to in compare or operate blocks.

- *%Cxx.D* - Counter xx done (Boolean, read only)
- *%Cxx.E* - Counter xx empty overflow (Boolean, read only)
- *%Cxx.F* - Counter xx full overflow (Boolean, read only)
- *%Cxx.V* - Counter xx current value (integer, read or write)
- *%Cxx.P* - Counter xx preset (integer, read or write)

14.2.6.6 COMPARE

For arithmetic comparison. Is variable *%XXX* = to this number (or evaluated number)

The compare block will be true when comparison is true. you can use most math symbols:

- +, -, *, /, = (standard math symbols)
- < (less than), > (greater than), <= (less or equal), >= (greater or equal), <> (not equal)
- (,) separate into groups example *%IF1=2,&%IF2<5* in pseudo code translates to if *%IF1* is equal to 2 and *%IF2* is less than 5 then the comparison is true. Note the comma separating the two groups of comparisons.
- ^ (exponent), % (modulus), & (and), | (or), .. -
- ABS (absolute), MOY (French for average), AVG (average)

For example *ABS(%W2)=1*, *MOY(%W1,%W2)<3*.

No spaces are allowed in the comparison equation. For example *%C0.V>%C0.P* is a valid comparison expression while *%C0.V > %C0.P* is not a valid expression.

There is a list of Variables down the page that can be used for reading from and writing to ladder objects. When a new compare block is opened be sure and delete the # symbol when you enter a compare.

To find out if word variable #1 is less than 2 times the current value of counter #0 the syntax would be:

```
%W1<2*%C0.V
```

To find out if S32in bit 2 is equal to 10 the syntax would be:

```
%IW2=10
```

Note: Compare uses the arithmetic equals not the double equals that programmers are used to.

14.2.6.7 VARIABLE ASSIGNMENT

For variable assignment, e.g. assign this number (or evaluated number) to this variable %xxx, there are two math functions MINI and MAXI that check a variable for maximum (0x80000000) and minimum values (0x07FFFFFFF) (think signed values) and keeps them from going beyond.

When a new variable assignment block is opened be sure to delete the # symbol when you enter an assignment.

To assign a value of 10 to the timer preset of IEC Timer 0 the syntax would be:

```
%TM0.P=10
```

To assign the value of 12 to s32out bit 3 the syntax would be:

```
%QW3=12
```

Note

When you assign a value to a variable with the variable assignment block the value is retained until you assign a new value using the variable assignment block. The last value assigned will be restored when LinuxCNC is started.

The following figure shows an Assignment and a Comparison Example. %QW0 is a S32out bit and %IW0 is a S32in bit. In this case the HAL pin classicladder.0.s32out-00 will be set to a value of 5 and when the HAL pin classicladder.0.s32in-00 is 0 the HAL pin classicladder.0.out-00 will be set to True.

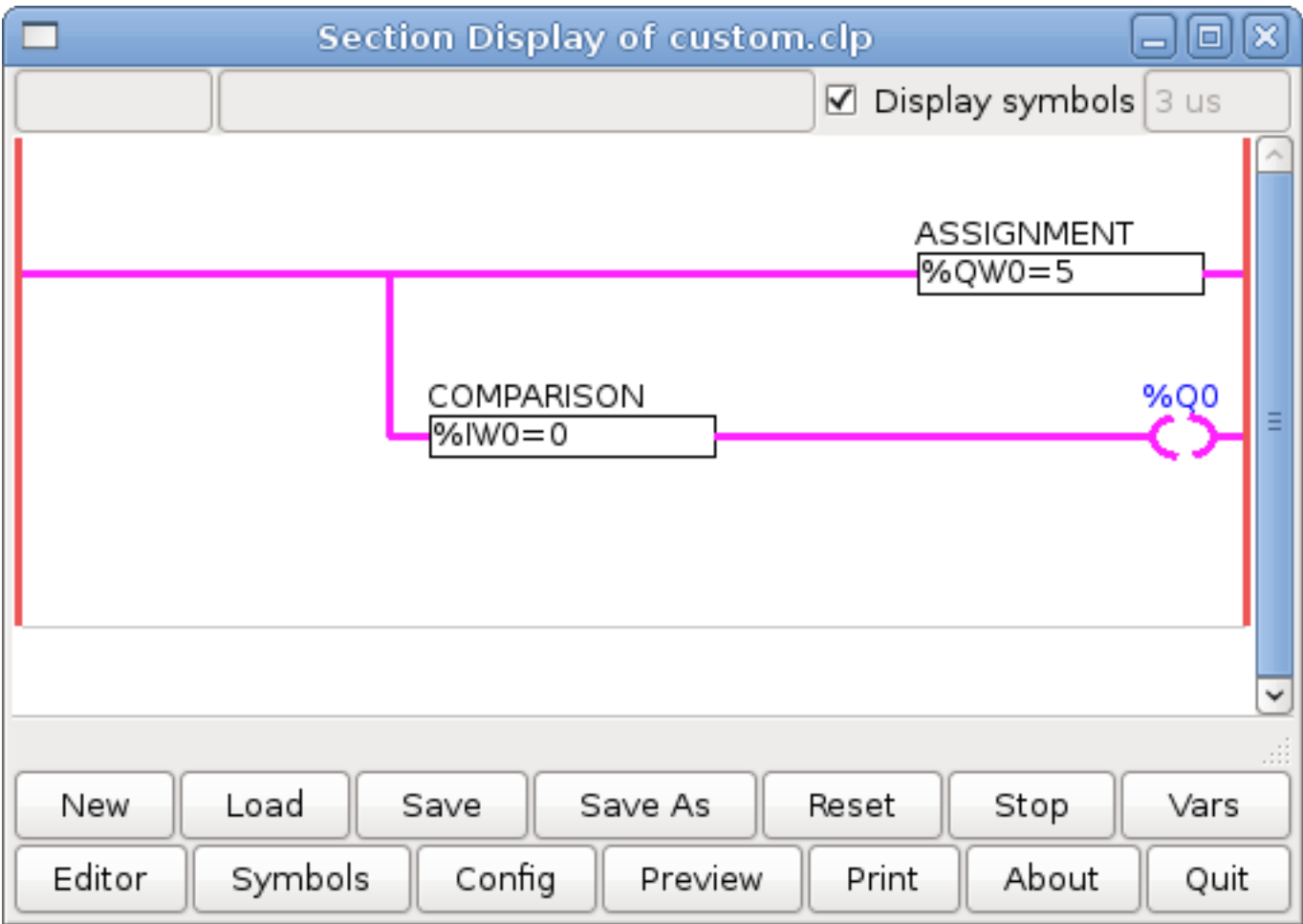


Figure 14.8: Assign/Compare Ladder Example

The screenshot shows a "Properties" dialog box. It has a title bar with a close button. Inside, there is a label "Expression" followed by a text input field containing "%QW0=5". Below this are three empty text input fields, each preceded by three dashes "---". At the bottom right is an "Apply" button.

Figure 14.9: Assignment Expression Example



Figure 14.10: Comparison Expression Example

14.2.6.8 COILS

Coils represent relay coils. They are controlled by the variable letter and number assigned to them. The variable letter can be B or Q and the number can be up to a three digit number eg. %Q3, or %B123. Q coils control HAL out pins, e.g. if %Q15 is energized then HAL pin classicladder.0.out-15 will be true. B coils are internal coils used to control program flow.

- *N.O. COIL* - (a relay coil.) When coil is energized it's N.O. contact will be closed (on, true, etc)
- *N.C. COIL* - (a relay coil that inverses its contacts.) When coil is energized it's N.O. contact will be open (off, false, etc)
- *SET COIL* - (a relay coil with latching contacts) When coil is energized it's N.O. contact will be latched closed.
- *RESET COIL* - (a relay coil with latching contacts) When coil is energized It's N.O. contact will be latched open.
- *JUMP COIL* - (a *goto* coil) when coil is energized ladder program jumps to a rung (in the CURRENT section) -jump points are designated by a rung label. (Add rung labels in the section display, top left label box)
- *CALL COIL* - (a *gosub* coil) when coil is energized program jumps to a subroutine section designated by a subroutine number -subroutines are designated SR0 to SR9 (designate them in the section manager)



Warning

If you use a N.C. contact with a N.C. coil the logic will work (when the coil is energized the contact will be closed) but that is really hard to follow!

A JUMP COIL is used to *JUMP* to another section, like a goto in BASIC programming language.

If you look at the top left of the sections display window you will see a small label box and a longer comment box beside it. Now go to Editor→Modify then go back to the little box, type in a name.

Go ahead and add a comment in the comment section. This label name is the name of this rung only and is used by the JUMP COIL to identify where to go.

When placing a JUMP COIL, add it in the rightmost position and change the label to the rung you want to JUMP to.

A CALL COIL is used to go to a subroutine section then return, like a gosub in BASIC programming language.

If you go to the sections manager window hit the add section button. You can name this section, select what language it will use (ladder or sequential), and select what type (main or subroutine).

Select a subroutine number (SR0 for example). An empty section will be displayed and you can build your subroutine.

When you've done that, go back to the section manager and click on the your main section (default name prog1).

Now you can add a CALL COIL to your program. CALL COILs are to be placed at the rightmost position in the rung.

Remember to change the label to the subroutine number you chose before.

14.2.7 ClassicLadder Variables

These Variables are used in COMPARE or OPERATE to get information about, or change specs of, ladder objects such as changing a counter preset, or seeing if a timer is done running.

List of variables :

- %Bxxx - Bit memory xxx (Boolean)
- %Wxxx - Word memory xxx (32 bits signed integer)
- %IWxxx - Word memory xxx (S32 in pin)
- %QWxxx - Word memory xxx (S32 out pin)
- %IFxx - Word memory xx (Float in pin) (**converted to S32 in ClassicLadder**)
- %QFxx - Word memory xx (Float out pin) (**converted to S32 in ClassicLadder**)
- %Txx.R - Timer xx running (Boolean, user read only)
- %Txx.D - Timer xx done (Boolean, user read only)
- %Txx.V - Timer xx current value (integer, user read only)
- %Txx.P - Timer xx preset (integer)
- %TMxxx.Q - Timer xxx done (Boolean, read write)
- %TMxxx.P - Timer xxx preset (integer, read write)
- %TMxxx.V - Timer xxx value (integer, read write)
- %Mxx.R - Monostable xx running (Boolean)
- %Mxx.V - Monostable xx current value (integer, user read only)
- %Mxx.P - Monostable xx preset (integer)
- %Cxx.D - Counter xx done (Boolean, user read only)
- %Cxx.E - Counter xx empty overflow (Boolean, user read only)
- %Cxx.F - Counter xx full overflow (Boolean, user read only)
- %Cxx.V - Counter xx current value (integer)

- `%Cxx.P` - Counter xx preset (integer)
- `%Ixxx` - Physical input xxx (Boolean) (HAL input bit)
- `%Qxxx` - Physical output xxx (Boolean) (HAL output bit)
- `%Xxxx` - Activity of step xxx (sequential language)
- `%Xxxx.V` - Time of activity in seconds of step xxx (sequential language)
- `%Exx` - Errors (Boolean, read write(will be overwritten))
- *Indexed or vectored variables* - These are variables indexed by another variable. Some might call this vectored variables. Example: `%W0[%W4]` => if `%W4` equals 23 it corresponds to `%W23`

14.2.8 GRAFCET (State Machine) Programming



Warning

This is probably the least used and most poorly understood feature of ClassicLadder. Sequential programming is used to make sure a series of ladder events always happen in a prescribed order. Sequential programs do not work alone. There is always a ladder program as well that controls the variables. Here are the basic rules governing sequential programs:

- Rule 1 : Initial situation - The initial situation is characterized by the initial steps which are by definition in the active state at the beginning of the operation. There shall be at least one initial step.
- Rule 2 : R2, Clearing of a transition - A transition is either enabled or disabled. It is said to be enabled when all immediately preceding steps linked to its corresponding transition symbol are active, otherwise it is disabled. A transition cannot be cleared unless it is enabled, and its associated transition condition is true.
- Rule 3 : R3, Evolution of active steps - The clearing of a transition simultaneously leads to the active state of the immediately following step(s) and to the inactive state of the immediately preceding step(s).
- Rule 4 : R4, Simultaneous clearing of transitions - All simultaneous cleared transitions are simultaneously cleared.
- Rule 5 : R5, Simultaneous activation and deactivation of a step - If during operation, a step is simultaneously activated and deactivated, priority is given to the activation.

This is the SEQUENTIAL editor window. (Starting from the top left):

Selector arrow , Eraser

Ordinary step , Initial (Starting) step

Transition , Step and Transition

Transition Link-Downside , Transition Link-Upside

Pass-through Link-Downside , Pass-through Link-Upside Jump

Link, Comment Box

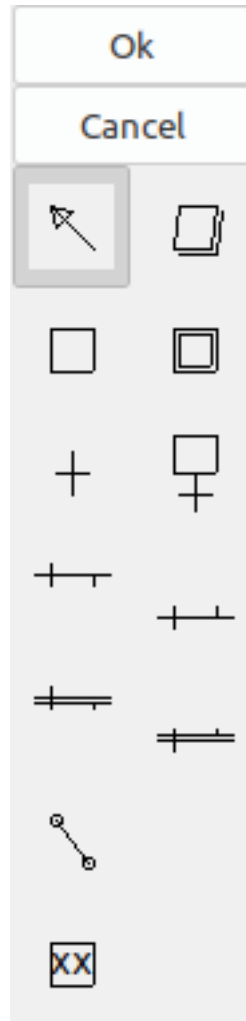


Figure 14.11: Sequence Editor Window

- *ORDINARY STEP* - has a unique number for each one
- *STARTING STEP* - a sequential program must have one. This is where the program will start.
- *TRANSITION* - This shows the variable that must be true for control to pass through to the next step.
- *STEP AND TRANSITION* - Combined for convenience
- *TRANSITION LINK-DOWNSIDE* - splits the logic flow to one of two possible lines based on which of the next steps is true first (Think OR logic)
- *TRANSITION LINK=UPSIDE* - combines two (OR) logic lines back in to one
- *PASS-THROUGH LINK-DOWNSIDE* - splits the logic flow to two lines that BOTH must be true to continue (Think AND logic)
- *PASS-THROUGH LINK-UPSIDE* - combines two concurrent (AND logic) logic lines back together
- *JUMP LINK* - connects steps that are not underneath each other such as connecting the last step to the first
- *COMMENT BOX* - used to add comments

To use links, you must have steps already placed. Select the type of link, then select the two steps or transactions one at a time. It takes practice!

With sequential programming: The variable %Xxxx (eg. %X5) is used to see if a step is active. The variable %Xxxx.V (eg. %X5.V) is used to see how long the step has been active. The %X and %X.v variables are use in LADDER logic. The variables assigned to the transitions (eg. %B) control whether the logic will pass to the next step. After a step has become active the transition variable that caused it to become active has no control of it anymore. The last step has to JUMP LINK back only to the beginning step.

14.2.9 Modbus

Things to consider:

- Modbus is a userspace program so it might have latency issues on a heavily laden computer.
- Modbus is not really suited to hard real time events such as position control of motors or to control E-stop.
- The ClassicLadder GUI must be running for Modbus to be running.
- Modbus is not fully finished so it does not do all modbus functions.

To get MODBUS to initialize you must specify that when loading the ClassicLadder userspace program.

Loading Modbus

```
loadusr -w classicladder --modmaster myprogram.clp
```

The -w makes HAL wait until you close ClassicLadder before closing realtime session. ClassicLadder also loads a TCP modbus slave if you add *--modserver* on command line.

Modbus Functions

- 1 - read coils
- 2 - read inputs
- 3 - read holding registers
- 4 - read input registers
- 5 - write single coils
- 6 - write single register
- 8 - echo test
- 15 - write multiple coils
- 16 - write multiple registers

If you do not specify a *-- modmaster* when loading the ClassicLadder user program this page will not be displayed.

Config

Period/object info	Modbus communication setup	Modbus I/O register setup 1	Modbus I/O register setup 2		
Slave Address	TypeAccess	1st Modbus Ele.	Nbr of Ele	Logic	1st I/Q/W Mapped
12	Read_INPUTS fnct- 2	1	1	<input type="checkbox"/> Inverted	1
12	Read_INPUTS fnct- 2	9	1	<input type="checkbox"/> Inverted	9
12	Write_COIL(S) fnct-5/15	0	1	<input type="checkbox"/> Inverted	0
	Read_REGS fnct- 4	1	1	<input type="checkbox"/> Inverted	0
	Write_REG(S) fnct-6/16	1	1	<input type="checkbox"/> Inverted	0
	Read_HOLD fnct- 3	1	1	<input type="checkbox"/> Inverted	0
	Slave_echo fnct- 8	1	1	<input type="checkbox"/> Inverted	0
	Read_INPUTS fnct- 2	1	1	<input type="checkbox"/> Inverted	0
	Read_INPUTS fnct- 2	1	1	<input type="checkbox"/> Inverted	0
	Read_INPUTS fnct- 2	1	1	<input type="checkbox"/> Inverted	0
	Read_INPUTS fnct- 2	1	1	<input type="checkbox"/> Inverted	0
	Read_INPUTS fnct- 2	1	1	<input type="checkbox"/> Inverted	0
	Read_INPUTS fnct- 2	1	1	<input type="checkbox"/> Inverted	0
	Read_INPUTS fnct- 2	1	1	<input type="checkbox"/> Inverted	0
	Read_INPUTS fnct- 2	1	1	<input type="checkbox"/> Inverted	0
	Read_INPUTS fnct- 2	1	1	<input type="checkbox"/> Inverted	0
	Read_INPUTS fnct- 2	1	1	<input type="checkbox"/> Inverted	0

Figure 14.12: Modbus I/O Config

Config

Period/object info | **Modbus communication setup** | Modbus I/O register setup

Serial port (blank = IP mode)

Serial baud rate

After transmit pause - milliseconds

After receive pause - milliseconds

Request Timeout length - milliseconds

Use RTS to send ☒ NO ☐ YES

Modbus element offset ☐ 0 ☒ 1

Debug level ☒ QUIET ☐ LEVEL 1 ☐ LEVEL 2 ☐ LEVEL 3

Read Coils/inputs map to ☒ %B ☐ %Q

Write Coils map from ☒ %B ☐ %Q ☐ %I

Read register/holding map to ☐ %W ☒ %QW

Write registers map from ☐ %W ☒ %QW ☐ %IW

Figure 14.13: Modbus Communication Config

- **SERIAL PORT** - For IP blank. For serial the location/name of serial driver eg. /dev/ttyS0 (or /dev/ttyUSB0 for a USB-to-serial converter).
- **SERIAL SPEED** - Should be set to speed the slave is set for - 300, 600, 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200 are supported.
- **PAUSE AFTER TRANSMIT** - Pause (milliseconds) after transmit and before receiving answer, some devices need more time (e.g., USB-to-serial converters).
- **PAUSE INTER-FRAME** - Pause (milliseconds) after receiving answer from slave. This sets the duty cycle of requests (it's a pause for EACH request).
- **REQUEST TIMEOUT LENGTH** - Length (milliseconds) of time before we decide that the slave didn't answer.
- **MODBUS ELEMENT OFFSET** - used to offset the element numbers by 1 (for manufacturers numbering differences).
- **DEBUG LEVEL** - Set this to 0-3 (0 to stop printing debug info besides no-response errors).
- **READ COILS/INPUTS MAP TO** - Select what variables that read coils/inputs will update. (B or Q).
- **WRITE COILS MAP TO** - Select what variables that write coils will updated from (B,Q,or I).
- **READ REGISTERS/HOLDING** - Select what variables that read registers will update (W or QW).
- **WRITE REGISTERS MAP TO** - Select what variables that read registers will updated from (W, QW, or IW).

- *SLAVE ADDRESS* - For serial the slaves ID number usually settable on the slave device (usually 1-256) For IP the slave IP address plus optionally the port number.
- *TYPE ACCESS* - This selects the MODBUS function code to send to the slave (eg what type of request).
- *COILS / INPUTS* - Inputs and Coils (bits) are read from/written to I, B, or Q variables (user selects).
- *REGISTERS (WORDS)* - Registers (Words/Numbers) map to IW, W, or QW variables (user selects).
- *1st MODBUS ELEMENT* - The address (or register number) of the first element in a group (remember to set MODBUS ELEMENT OFFSET properly).
- *NUMBER OF ELEMENTS* - The number of elements in this group.
- *LOGIC* - You can invert the logic here.
- *1st%I%Q IQ WQ MAPPED* - This is the starting number of %B, %I, %Q, %W, %IW, or %QW variables that are mapped onto/from the modbus element group (starting at the first modbus element number).

In the example above: Port number - for my computer /dev/ttyS0 was my serial port.

The serial speed is set to 9600 baud.

Slave address is set to 12 (on my VFD I can set this from 1-31, meaning I can talk to 31 VFDs maximum on one system).

The first line is set up for 8 input bits starting at the first register number (register 1). So register numbers 1-8 are mapped onto ClassicLadder's %B variables starting at %B1 and ending at %B8.

The second line is set for 2 output bits starting at the ninth register number (register 9) so register numbers 9-10 are mapped onto ClassicLadder's %Q variables starting at %Q9 ending at %Q10.

The third line is set to write 2 registers (16 bits each) starting at the 0th register number (register 0) so register numbers 0-1 are mapped onto ClassicLadder's %W variables starting at %W0 ending at %W1.

It's easy to make an off-by-one error as sometimes the modbus elements are referenced starting at one rather than 0 (actually by the standard that is the way it's supposed to be!) You can use the modbus element offset radio button to help with this.

The documents for your modbus slave device will tell you how the registers are set up- there is no standard way.

The SERIAL PORT, PORT SPEED, PAUSE, and DEBUG level are editable for changes (when you close the config window values are applied, though Radio buttons apply immediately).

To use the echo function select the echo function and add the slave number you wish to test. You don't need to specify any variables.

The number 257 will be sent to the slave number you specified and the slave should send it back. you will need to have ClassicLadder running in a terminal to see the message.

14.2.10 MODBUS Settings

Serial:

- ClassicLadder uses RTU protocol (not ASCII).
- 8 data bits, No parity is used, and 1 stop bit is also known as 8-N-1.
- Baud rate must be the same for slave and master. ClassicLadder can only have one baud rate so all the slaves must be set to the same rate.
- Pause inter frame is the time to pause after receiving an answer.
- MODBUS_TIME_AFTER_TRANSMIT is the length of pause after sending a request and before receiving an answer (this apparently helps with USB converters which are slow).

14.2.10.1 MODBUS Info

- ClassicLadder can use distributed inputs/outputs on modules using the modbus protocol ("master": polling slaves).
- The slaves and theirs I/O can be configured in the config window.
- 2 exclusive modes are available : ethernet using Modbus/TCP and serial using Modbus/RTU.
- No parity is used.
- If no port name for serial is set, TCP/IP mode will be used...
- The slave address is the slave address (Modbus/RTU) or the IP address.
- The IP address can be followed per the port number to use (xx.xx.xx.xx:pppp) else the port 9502 will be used per default.
- 2 products have been used for tests: a Modbus/TCP one (Adam-6051, <http://www.advantech.com>) and a serial Modbus/RTU one (<http://www.ipac.ws>).
- See examples: adam-6051 and modbus_rtu_serial.
- Web links: <http://www.modbus.org> and this interesting one: <http://www.iatips.com/modbus.html>
- MODBUS TCP SERVER INCLUDED
- ClassicLadder has a Modbus/TCP server integrated. Default port is 9502. (the previous standard 502 requires that the application must be launched with root privileges).
- List of Modbus functions code supported are: 1, 2, 3, 4, 5, 6, 15 and 16.
- Modbus bits and words correspondence table is actually not parametric and correspond directly to the %B and %W variables.

More information on modbus protocol is available on the internet.

<http://www.modbus.org/>

14.2.10.2 Communication Errors

If there is a communication error, a warning window will pop up (if the GUI is running) and %E0 will be true. Modbus will continue to try to communicate. The %E0 could be used to make a decision based on the error. A timer could be used to stop the machine if timed out, etc.

14.2.11 Debugging modbus problems

A good reference for the protocol: http://www.modbus.org/docs/Modbus_Application_Protocol_V1_1b.pdf
If you run linuxcnc/classocladder from a terminal, it will print the Modbus commands and slave responses.

Here we set ClassicLadder to request slave 1, to read holding registers (function code 3) starting at address 8448 (0x2100). We ask for 1 (2 byte wide) data element to be returned. We map it to a ClassicLadder variable starting at 2.

Period/object info		Modbus communication setup		Modbus I/O register setup		
Slave Address	Request Type		1st Modbus Ele.	# of Ele	Logic	1st Variable mapped
1	Read_HOLD_REG	fnctn- 3	8448	1	<input type="checkbox"/> Inverted	2
	Read_discrete_INPUTS	fnctn- 2		1	<input type="checkbox"/> Inverted	0
	Read_discrete_INPUTS	fnctn- 2	1	1	<input type="checkbox"/> Inverted	0
	Read_discrete_INPUTS	fnctn- 2	1	1	<input type="checkbox"/> Inverted	0
	Read_discrete_INPUTS	fnctn- 2	1	1	<input type="checkbox"/> Inverted	0
	Read_discrete_INPUTS	fnctn- 2	1	1	<input type="checkbox"/> Inverted	0
	Read_discrete_INPUTS	fnctn- 2	1	1	<input type="checkbox"/> Inverted	0
	Read_discrete_INPUTS	fnctn- 2	1	1	<input type="checkbox"/> Inverted	0
	Read_discrete_INPUTS	fnctn- 2	1	1	<input type="checkbox"/> Inverted	0
	Read_discrete_INPUTS	fnctn- 2	1	1	<input type="checkbox"/> Inverted	0
	Read_discrete_INPUTS	fnctn- 2	1	1	<input type="checkbox"/> Inverted	0
	Read_discrete_INPUTS	fnctn- 2	1	1	<input type="checkbox"/> Inverted	0
	Read_discrete_INPUTS	fnctn- 2	1	1	<input type="checkbox"/> Inverted	0
	Read_discrete_INPUTS	fnctn- 2	1	1	<input type="checkbox"/> Inverted	0
	Read_discrete_INPUTS	fnctn- 2	1	1	<input type="checkbox"/> Inverted	0
	Read_discrete_INPUTS	fnctn- 2	1	1	<input type="checkbox"/> Inverted	0

Figure 14.14: Modbus I/O Register Setup

Note in this image we have set the debug level to 1 so modbus messages are printed to the terminal. We have mapped our read and written holding registers to ClassicLadder's %W variables so our returned data will be in %W2 as in the other image we mapped the data starting at the 2nd element.

Period/object info

Modbus communication setup

Modbus I/O register setup

Serial port (blank = IP mode)

/dev/ttyS0

Serial baud rate

115200

After transmit pause - milliseconds

0

After receive pause - milliseconds

200

Request Timeout length - milliseconds

500

Use RTS to send

☒ NO ☐ YES

Modbus element offset

☒ 0 ☐ 1

Debug level

☐ QUIET ☒ LEVEL 1 ☐ LEVEL 2 ☐ LEVEL 3

Read Coils/inputs map to

☒ %B ☐ %Q

Write Coils map from

☒ %B ☐ %Q ☐ %I

Read register/holding map to

☒ %W ☐ %QW

Write registers map from

☒ %W ☐ %QW ☐ %IW

Figure 14.15: Modbus Communication Setup

14.2.11.1 Request

Lets look at an example of reading one hold register at 8448 Decimal (0x2100 Hex).
Looking in the Modbus protocol reference:

Table 14.2: Read holding register request

Name	number of bytes	Value (hex)
Function code	(1 Byte)	3 (0x03)
Starting Address	(2 Bytes)	0 - 65535 (0x0000 to 0xFFFF)
Number of Registers	(2 Bytes)	1 to 125 (0x7D)
Checksum	(2 bytes)	Calculated automatically

Here is an example sent command as printed in the terminal (all Hex):

```
INFO CLASSICLADDER- Modbus I/O module to send: Lgt=8 <- Slave address-1 Function code-3 ↵
Data-21 0 0 1 8E 36
```

Meaning (Hex):

- Lgt = 8 = message is 8 bytes long including slave number and checksum number
- Slave number = 1 (0x1) = Slave address 1
- Function code = 3 (0x3) = read holding register
- Start at address = highbyte 33 (0x21) lowbyte 0 (0x00) = combined address = 8448 (0x2100)
- Number of Registers = 1 (0x1) = return 1 2-byte register (holding and reading registers are always 2 bytes wide)
- Checksum = high byte 0x8E lowbyte 0x36 = (0x8E36)

14.2.11.2 Error response

If there is an error response, it sends the function code plus 0x80, an error code, and a checksum. Getting an error response means the slave is seeing the request command but can not give valid data. Looking in the Modbus protocol reference:

Table 14.3: Error returned for function code 3 (read holding register)

Name	Number of bytes	Value (hex)
Error code	1 Byte	131 (0x83)
Exception code	1 Byte	1-4 (0x01 to 0x04)
Checksum	(2 bytes)	Calculated automatically

Exception code meaning:

- 1 - illegal Function
- 2 - illegal data address
- 3 - illegal data value
- 4 - slave device failure

Here is an example received command as printed in the terminal (all Hex):


```
INFO CLASSICLADDER- Modbus I/O module received: Lgt=5 -> (Slave address-1 Function code-83 ) 2 C0 F1 ↵
```

Meaning (Hex):

- Slave number = 1 (0x1) = Slave address 1
- Function code = 131 (0x83) = error while reading holding register
- Error code = 2 (0x2) = illegal data address requested
- Checksum = (0x8E36)

14.2.11.3 Data response

Looking in the Modbus protocol reference for Response:

Table 14.4: Data response for function code 3 (read holding register)

Name	number of bytes	Value (hex)
Function code	1 Byte	3 (0x03)
Byte count	1 Byte	2 x N*
Register value	N* x 2 Bytes	returned value of requested address
Checksum	(2 bytes)	calculated automatically

*N = Number of registers

Here is an example received command as printed in the terminal (all Hex):

```
INFO CLASSICLADDER- Modbus I/O module received: Lgt=7 -> (Slave address-1 Function code-3 2 0 0 B8 44) ↵
```

meaning (Hex):

- Slave number = 1 (0x1) = Slave address 1
- Requested function code = 3 (0x3) = read holding register requested
- count of byte registers = 2 (0x1) = return 2 bytes (each register value is 2 bytes wide)
- value of highbyte = 0 (0x0) = high byte value of address 8448 (0x2100)
- value of lowbyte = 0 (0x0) = high byte value of address 8448 (0x2100)
- Checksum = (0xB844)

(high and low bytes are combined to create a 16 bit value and then transferred to ClassicLadder's variable.) Read Registers can be mapped to %W or %QW (internal memory or HAL out pins). Write registers can be mapped from %W, %QW or %IW (internal memory, HAL out pins or HAL in pins). The variable number will start at the number entered in the modbus I/O registry setup page's column: *First variable mapped*. If multiple registers are requested in one read/write then the variable number are sequential after the first one.

14.2.11.4 MODBUS Bugs

- In compare blocks the function $\%W = \text{ABS}(\%W1 - \%W2)$ is accepted but does not compute properly. only $\%W0 = \text{ABS}(\%W1)$ is currently legal.
- When loading a ladder program it will load Modbus info but will not tell ClassicLadder to initialize Modbus. You must initialize Modbus when you first load the GUI by adding `--modmaster`.
- If the section manager is placed on top of the section display, across the scroll bar and exit is clicked the user program crashes.
- When using `--modmaster` you must load the ladder program at the same time or else only TCP will work.
- reading/writing multiple registers in Modbus has checksum errors.

14.2.12 Setting up ClassicLadder

In this section we will cover the steps needed to add ClassicLadder to a Stepconf Wizard generated config. On the advanced Configuration Options page of Stepconf Wizard check off "Include ClassicLadder PLC".

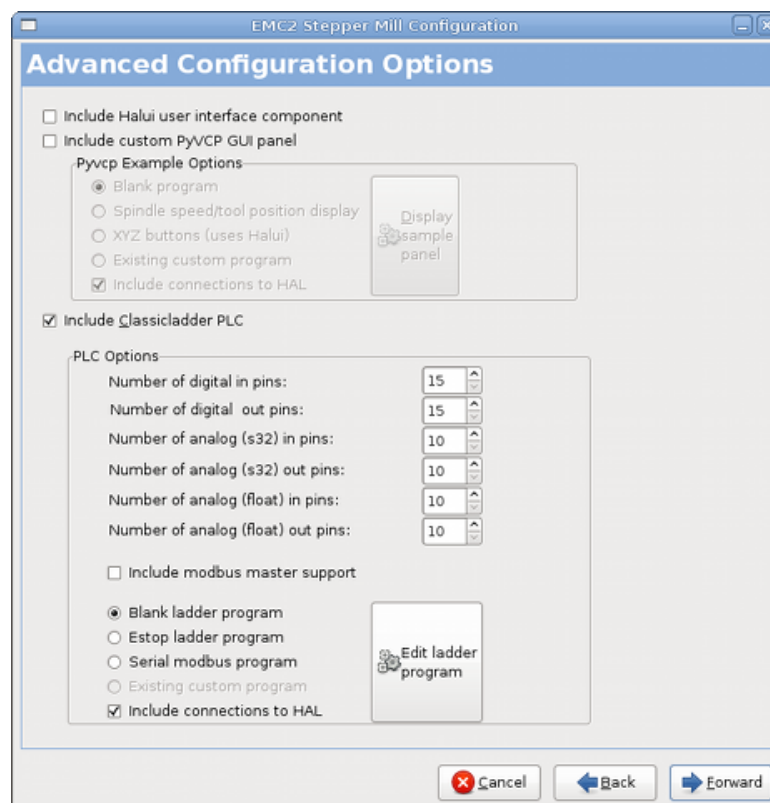


Figure 14.16: Stepconf ClassicLadder

14.2.12.1 Add the Modules

If you used the Stepconf Wizard to add ClassicLadder you can skip this step.

To manually add ClassicLadder you must first add the modules. This is done by adding a couple of lines to the custom.hal file.

This line loads the real time module:

```
loadrt classicladder_rt
```

This line adds the ClassicLadder function to the servo thread:

```
addf classicladder.0.refresh servo-thread
```

14.2.12.2 Adding Ladder Logic

Now start up your config and select "File/Ladder Editor" to open up the ClassicLadder GUI. You should see a blank Section Display and Sections Manager window as shown above. In the Section Display window open the Editor. In the Editor window select Modify. Now a Properties window pops up and the Section Display shows a grid. The grid is one rung of ladder. The rung can contain branches. A simple rung has one input, a connector line and one output. A rung can have up to six horizontal branches. While it is possible to have more than one circuit in a run the results are not predictable.

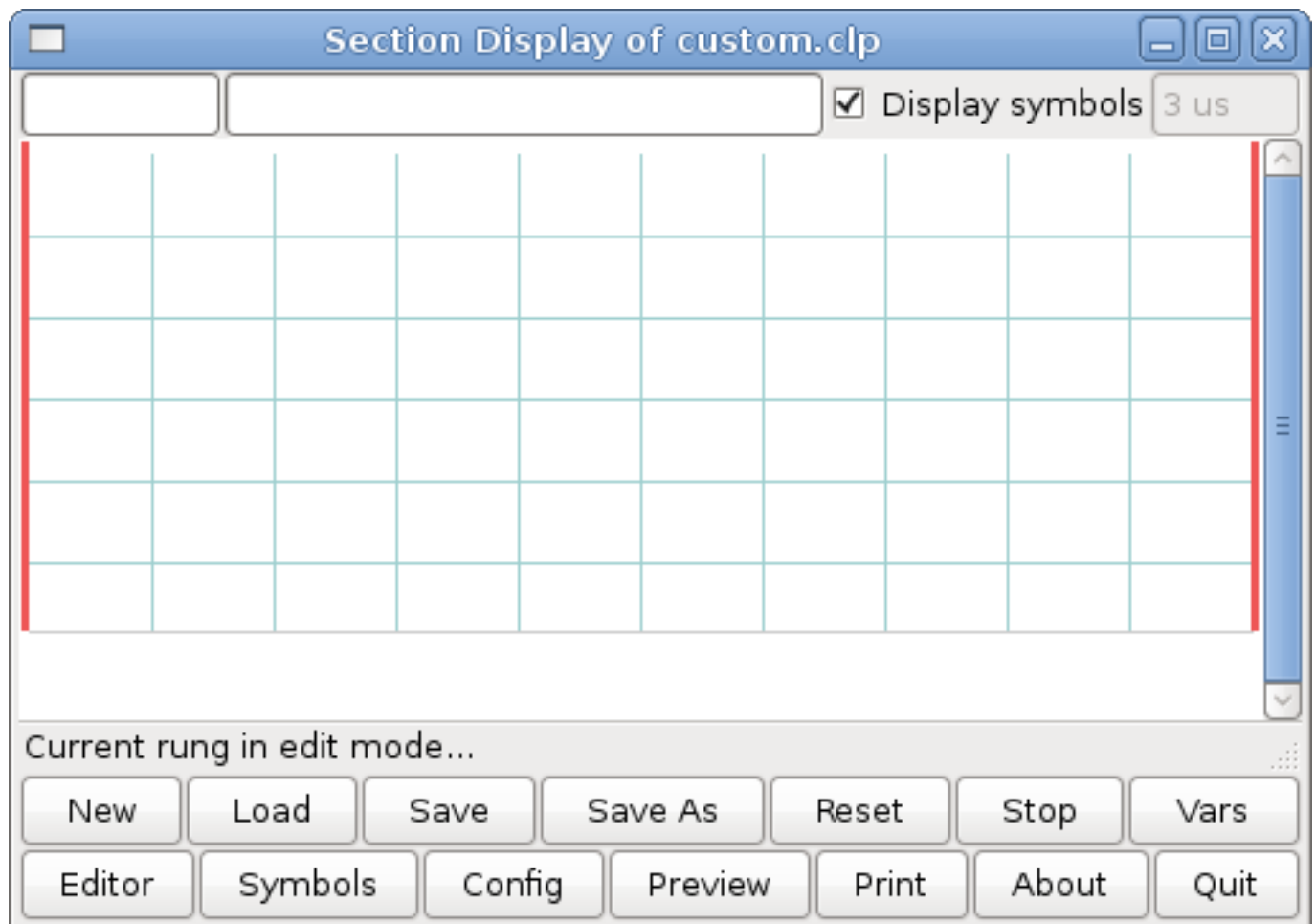


Figure 14.17: Section Display with Grid

Now click on the N.O. Input in the Editor Window.

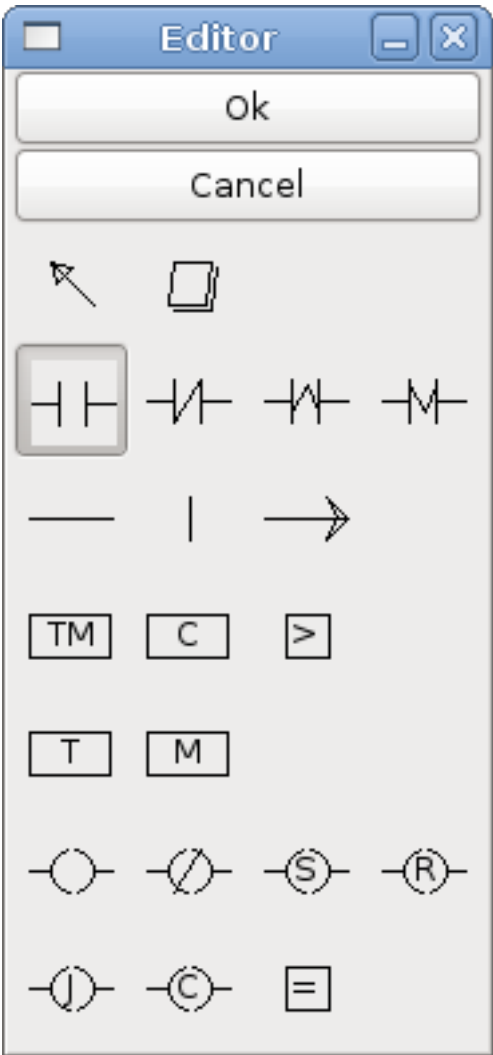


Figure 14.18: Editor Window

Now click in the upper left grid to place the N.O. Input into the ladder.



Figure 14.19: Section Display with Input

Repeat the above steps to add a N.O. Output to the upper right grid and use the Horizontal Connection to connect the two. It should look like the following. If not, use the Eraser to remove unwanted sections.



Figure 14.20: Section Display with Rung

Now click on the OK button in the Editor window. Now your Section Display should look like this:



Figure 14.21: Section Display Finished

To save the new file select *Save As* and give it a name. The *.clp* extension will be added automatically. It should default to the running config directory as the place to save it.

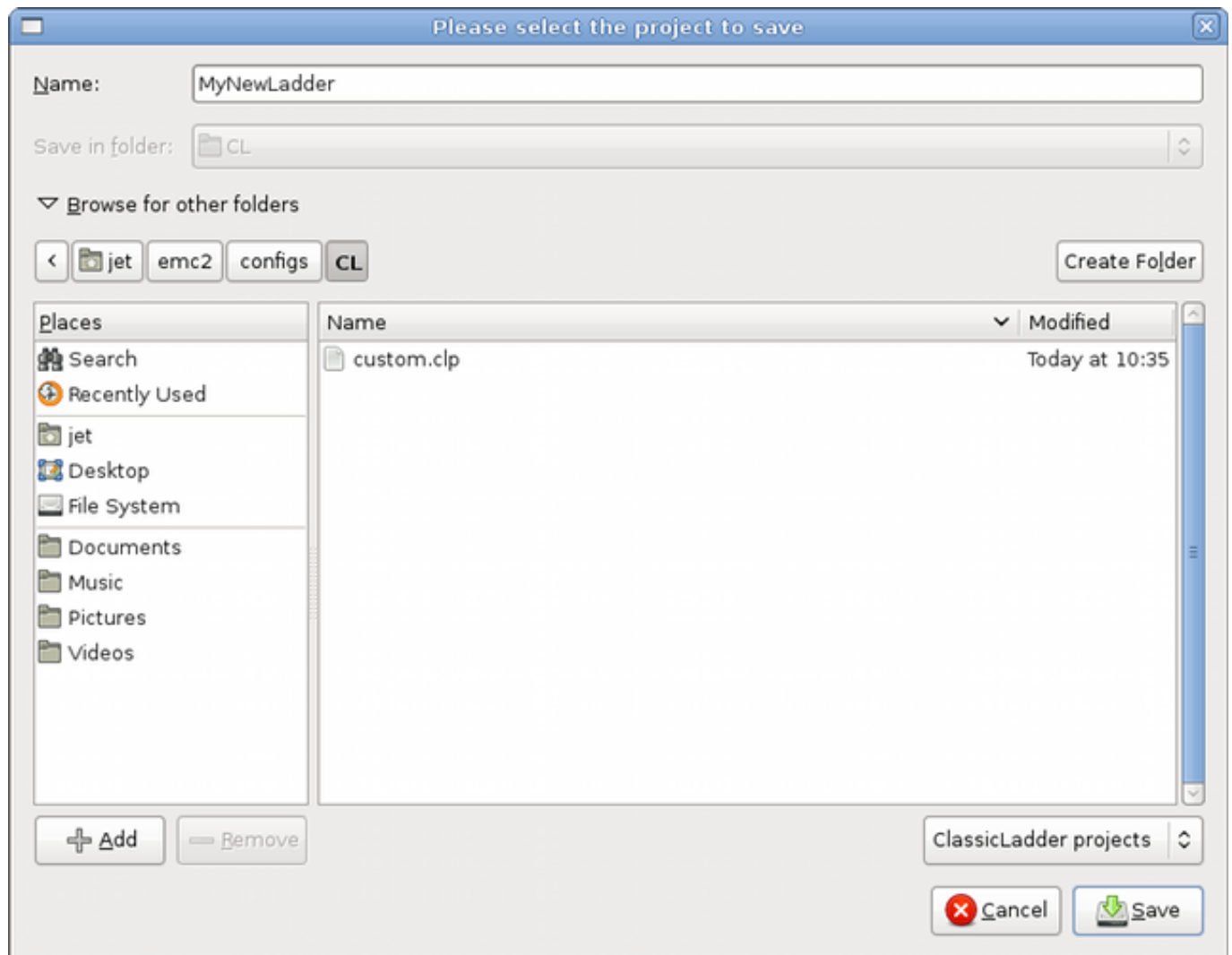


Figure 14.22: Save As Dialog

Again if you used the Stepconf Wizard to add ClassicLadder you can skip this step.

To manually add a ladder you need to add a line to your custom.hal file that will load your ladder file. Close your LinuxCNC session and add this line to your custom.hal file.

```
loadusr -w classicladder --nogui MyLadder.clp
```

Now if you start up your LinuxCNC config your ladder program will be running as well. If you select "File/Ladder Editor", the program you created will show up in the Section Display window.

14.3 ClassicLadder Examples

14.3.1 Wrapping Counter

To have a counter that *wraps around* you have to use the preset pin and the reset pin. When you create the counter set the preset at the number you wish to reach before wrapping around to 0. The logic is if the counter value is over the preset then reset the counter and if the underflow is on then

set the counter value to the preset value. As you can see in the example when the counter value is greater than the counter preset the counter reset is triggered and the value is now 0. The underflow output %Q2 will set the counter value at the preset when counting backwards.

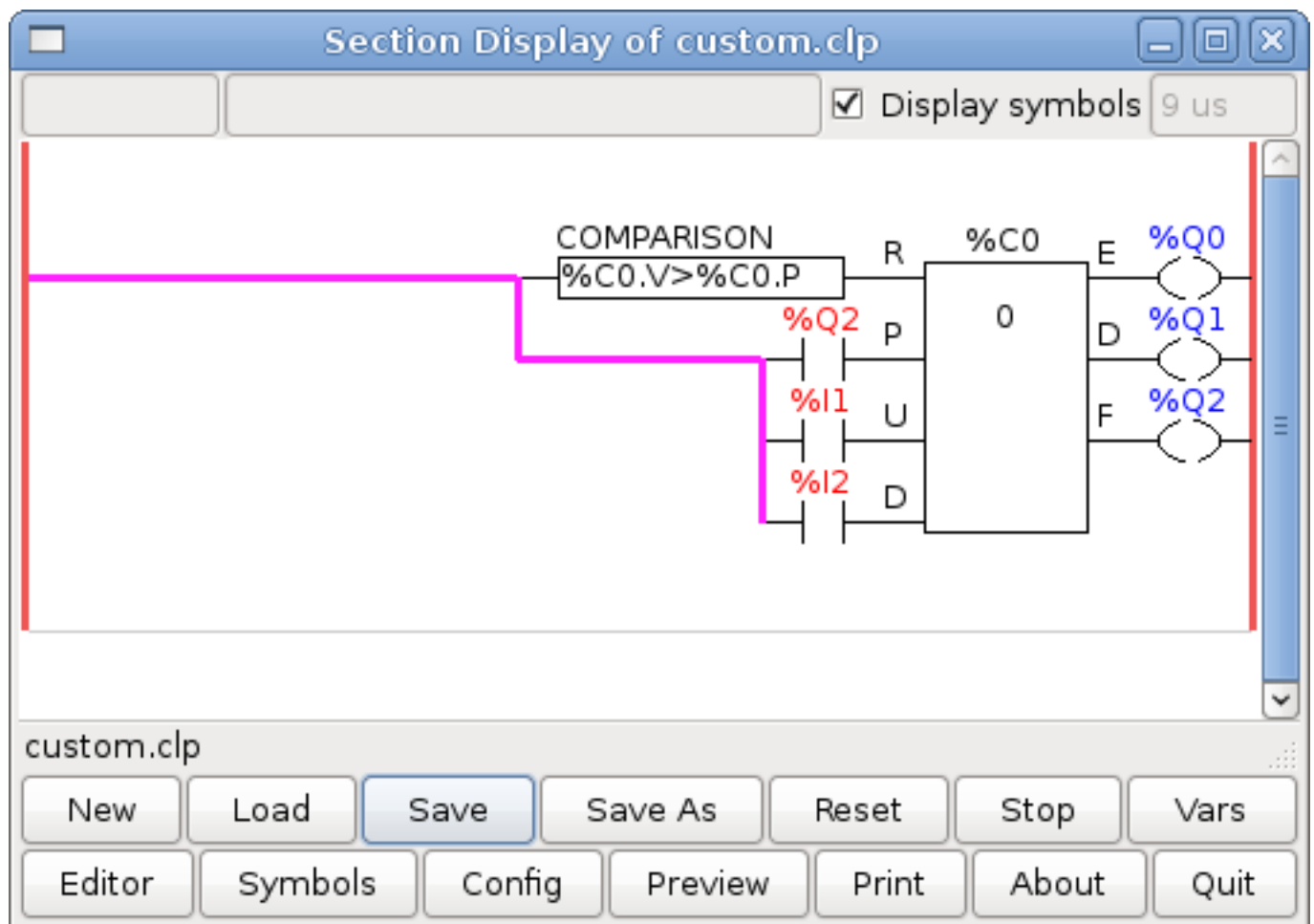


Figure 14.23: Wrapping Counter

14.3.2 Reject Extra Pulses

This example shows you how to reject extra pulses from an input. Suppose the input pulse %I0 has an annoying habit of giving an extra pulse that spoils our logic. The TOF (Timer Off Delay) prevents the extra pulse from reaching our cleaned up output %Q0. How this works is when the timer gets an input the output of the timer is on for the duration of the time setting. Using a normally closed contact %TM0.Q the output of the timer blocks any further inputs from reaching our output until it times out.

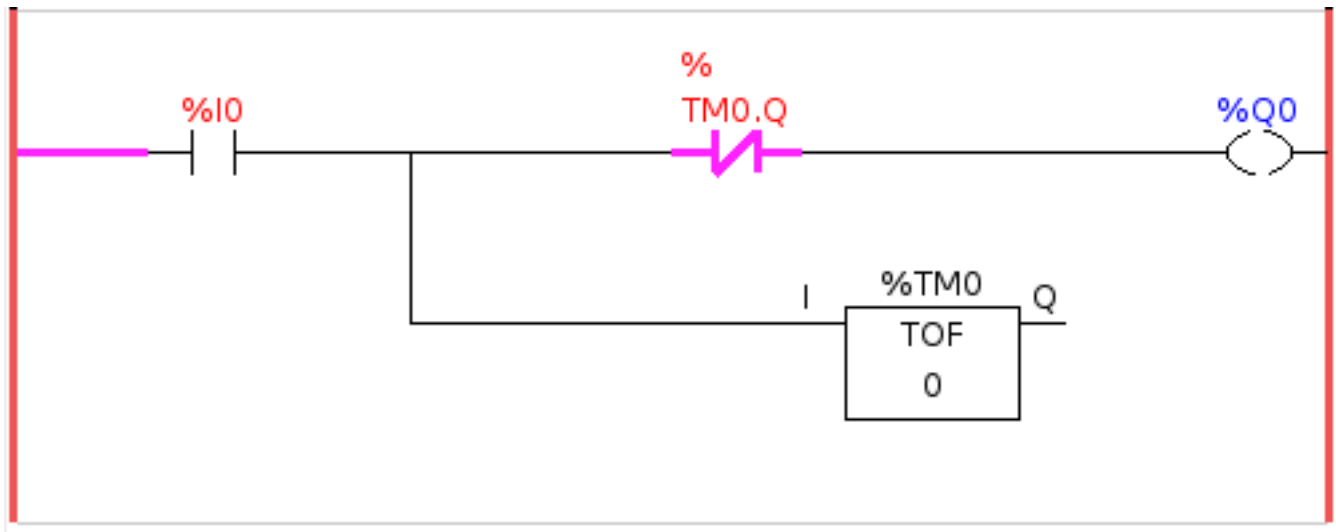


Figure 14.24: Reject Extra Pulse

14.3.3 External E-Stop

The External E-Stop example is in the `/config/classicladder/cl-estop` folder. It uses a PyVCP panel to simulate the external components.

To interface an external E-Stop to LinuxCNC and have the external E-Stop work together with the internal E-Stop requires a couple of connections through ClassicLadder.

First we have to open the E-Stop loop in the main HAL file by commenting out by adding the pound sign as shown or removing the following lines.

```
# net estop-out <= iocontrol.0.user-enable-out
# net estop-out => iocontrol.0.emc-enable-in
```

Next we add ClassicLadder to our custom.hal file by adding these two lines:

```
loadrt classicladder_rt
addf classicladder.0.refresh servo-thread
```

Next we run our config and build the ladder as shown here.



Figure 14.25: E-Stop Section Display

After building the ladder select Save As and save the ladder as estop.clp

Now add the following line to your custom.hal file.

```
# Load the ladder
loadusr classicladder --nogui estop.clp
```

I/O assignments

- %I0 = Input from the PyVCP panel simulated E-Stop (the checkbox)
- %I1 = Input from LinuxCNC's E-Stop
- %I2 = Input from LinuxCNC's E-Stop Reset Pulse
- %I3 = Input from the PyVCP panel reset button
- %Q0 = Output to LinuxCNC to enable
- %Q1 = Output to external driver board enable pin (use a N/C output if your board had a disable pin)

Next we add the following lines to the custom_postgui.hal file

```
# E-Stop example using PyVCP buttons to simulate external components

# The PyVCP checkbutton simulates a normally closed external E-Stop
net ext-estop classicladder.0.in-00 <= pyvcp.py-estop

# Request E-Stop Enable from LinuxCNC
net estop-all-ok iocontrol.0.emc-enable-in <= classicladder.0.out-00

# Request E-Stop Enable from PyVCP or external source
net ext-estop-reset classicladder.0.in-03 <= pyvcp.py-reset

# This line resets the E-Stop from LinuxCNC
net emc-reset-estop iocontrol.0.user-request-enable => classicladder.0.in-02

# This line enables LinuxCNC to unlatch the E-Stop in ClassicLadder
net emc-estop iocontrol.0.user-enable-out => classicladder.0.in-01

# This line turns on the green indicator when out of E-Stop
net estop-all-ok => pyvcp.py-es-status
```

Next we add the following lines to the panel.xml file. Note you have to open it with the text editor not the default html viewer.

```
<pyvcp>
<vbox>
<label><text>"E-Stop Demo"</text></label>
<led>
<halpin>"py-es-status"</halpin>
<size>50</size>
<on_color>"green"</on_color>
<off_color>"red"</off_color>
</led>
<checkbutton>
<halpin>"py-estop"</halpin>
<text>"E-Stop"</text>
</checkbutton>
</vbox>
<button>
<halpin>"py-reset"</halpin>
<text>"Reset"</text>
</button>
</pyvcp>
```

Now start up your config and it should look like this.



Figure 14.26: AXIS E-Stop

Note that in this example like in real life you must clear the remote E-Stop (simulated by the checkbox) before the AXIS E-Stop or the external Reset will put you in OFF mode. If the E-Stop in the AXIS screen was pressed, you must press it again to clear it. You cannot reset from the external after you do an E-Stop in AXIS.

14.3.4 Timer/Operate Example

In this example we are using the Operate block to assign a value to the timer preset based on if an input is on or off.

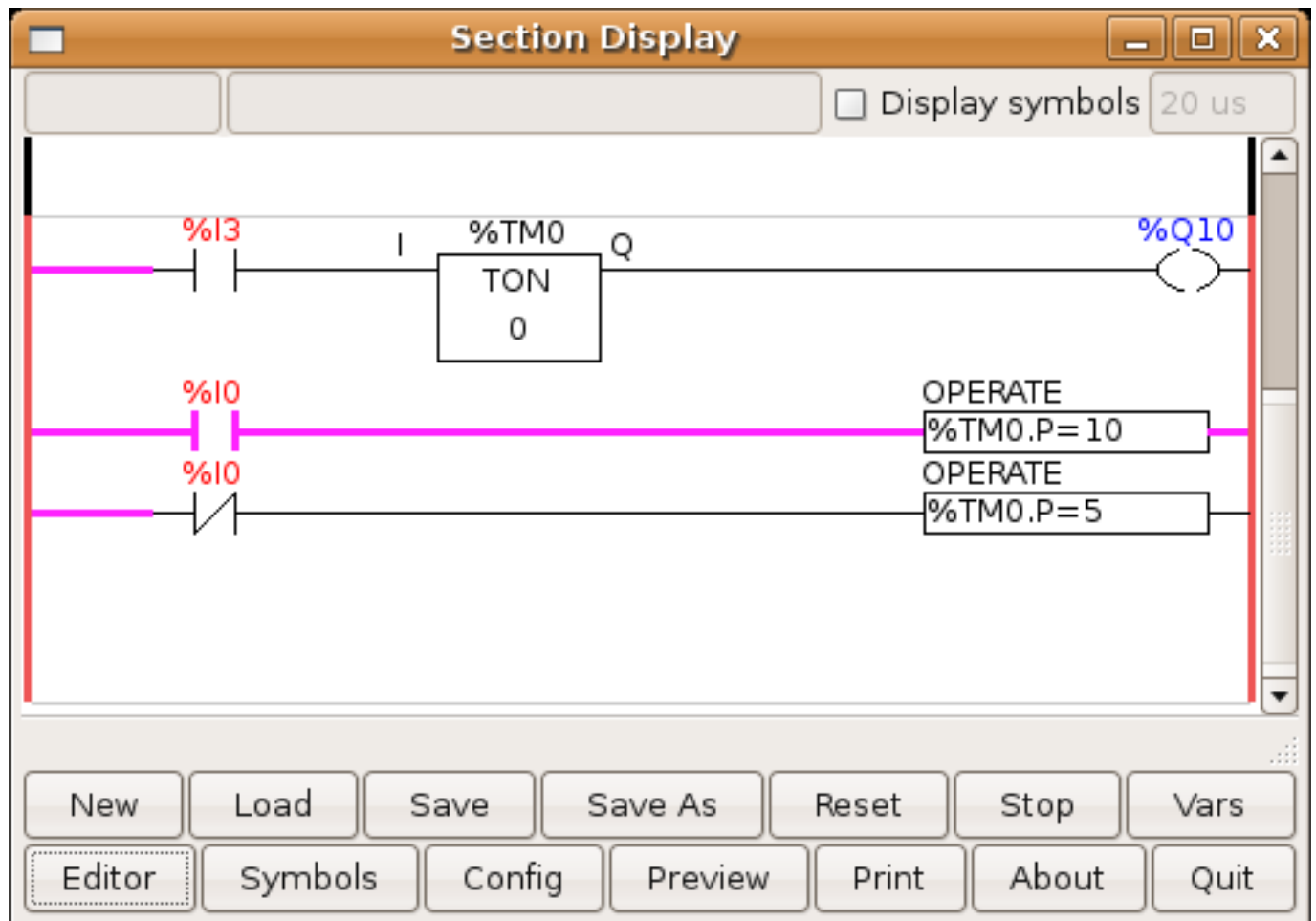


Figure 14.27: Timer/Operate Example

In this case %I0 is true so the timer preset value is 10. If %I0 was false the timer preset would be 5.

Chapter 15

HAL

15.1 HAL Introduction

HAL stands for Hardware Abstraction Layer. At the highest level, it is simply a way to allow a number of *building blocks* to be loaded and interconnected to assemble a complex system. The *Hardware* part is because HAL was originally designed to make it easier to configure LinuxCNC for a wide variety of hardware devices. Many of the building blocks are drivers for hardware devices. However, HAL can do more than just configure hardware drivers.

15.1.1 HAL System Design

HAL is based on traditional system design techniques.

HAL is based on the same principles that are used to design hardware circuits and systems, so it is useful to examine those principles first.

Any system, including a CNC machine, consists of interconnected components. For the CNC machine, those components might be the main controller, servo amps or stepper drives, motors, encoders, limit switches, pushbutton pendants, perhaps a VFD for the spindle drive, a PLC to run a toolchanger, etc. The machine builder must select, mount and wire these pieces together to make a complete system.

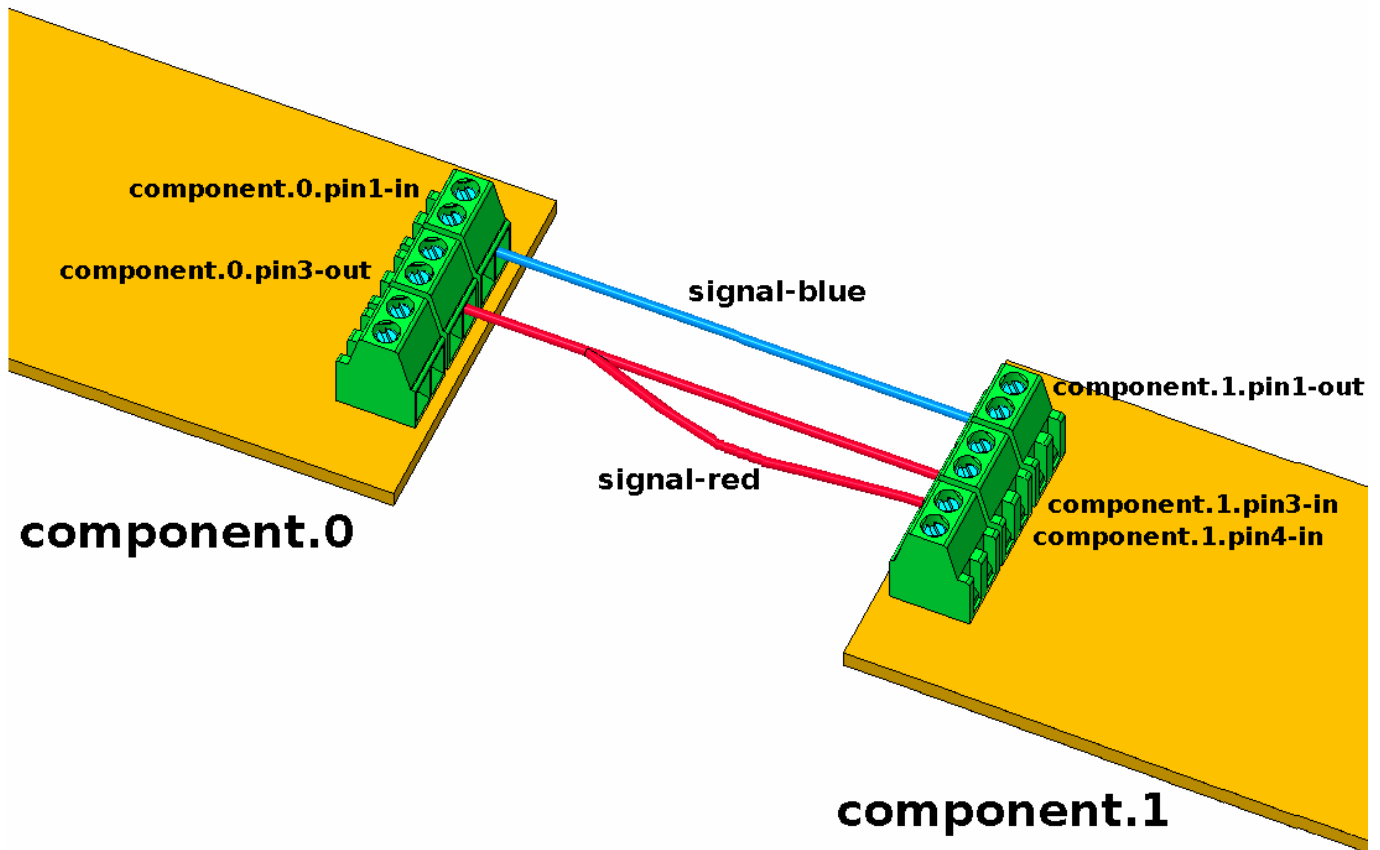


Figure 15.1: HAL Concept - Connecting like electrical circuits.

Figure one would be written in HAL code like this:

net signal-blue	component.0.pin1-in	component.1.pin1-out	
net signal-red	component.0.pin3-out	component.1.pin3-in	component.1.pin4-in

15.1.1.1 Part Selection

The machine builder does not need to worry about how each individual part works. He treats them as black boxes. During the design stage, he decides which parts he is going to use - steppers or servos, which brand of servo amp, what kind of limit switches and how many, etc. The integrator's decisions about which specific components to use is based on what that component does and the specifications supplied by the manufacturer of the device. The size of a motor and the load it must drive will affect the choice of amplifier needed to run it. The choice of amplifier may affect the kinds of feedback needed by the amp and the velocity or position signals that must be sent to the amp from a control.

In the HAL world, the integrator must decide what HAL components are needed. Usually every interface card will require a driver. Additional components may be needed for software generation of step pulses, PLC functionality, and a wide variety of other tasks.

15.1.1.2 Interconnection Design

The designer of a hardware system not only selects the parts, he also decides how those parts will be interconnected. Each black box has terminals, perhaps only two for a simple switch, or dozens for a servo drive or PLC. They need to be wired together. The motors connect to the servo amps, the limit switches connect to the controller, and so on. As the machine builder works on the design, he creates a large wiring diagram that shows how all the parts should be interconnected.

When using HAL, components are interconnected by signals. The designer must decide which signals are needed, and what they should connect.

15.1.1.3 Implementation

Once the wiring diagram is complete it is time to build the machine. The pieces need to be acquired and mounted, and then they are interconnected according to the wiring diagram. In a physical system, each interconnection is a piece of wire that needs to be cut and connected to the appropriate terminals.

HAL provides a number of tools to help *build* a HAL system. Some of the tools allow you to *connect* (or disconnect) a single *wire*. Other tools allow you to save a complete list of all the parts, wires, and other information about the system, so that it can be *rebuilt* with a single command.

15.1.1.4 Testing

Very few machines work right the first time. While testing, the builder may use a meter to see whether a limit switch is working or to measure the DC voltage going to a servo motor. He may hook up an oscilloscope to check the tuning of a drive, or to look for electrical noise. He may find a problem that requires the wiring diagram to be changed; perhaps a part needs to be connected differently or replaced with something completely different.

HAL provides the software equivalents of a voltmeter, oscilloscope, signal generator, and other tools needed for testing and tuning a system. The same commands used to build the system can be used to make changes as needed.

15.1.1.5 Summary

This document is aimed at people who already know how to do this kind of hardware system integration, but who do not know how to connect the hardware to LinuxCNC. See the [Remote Start Example](#) section in the HAL UI Examples documentation.

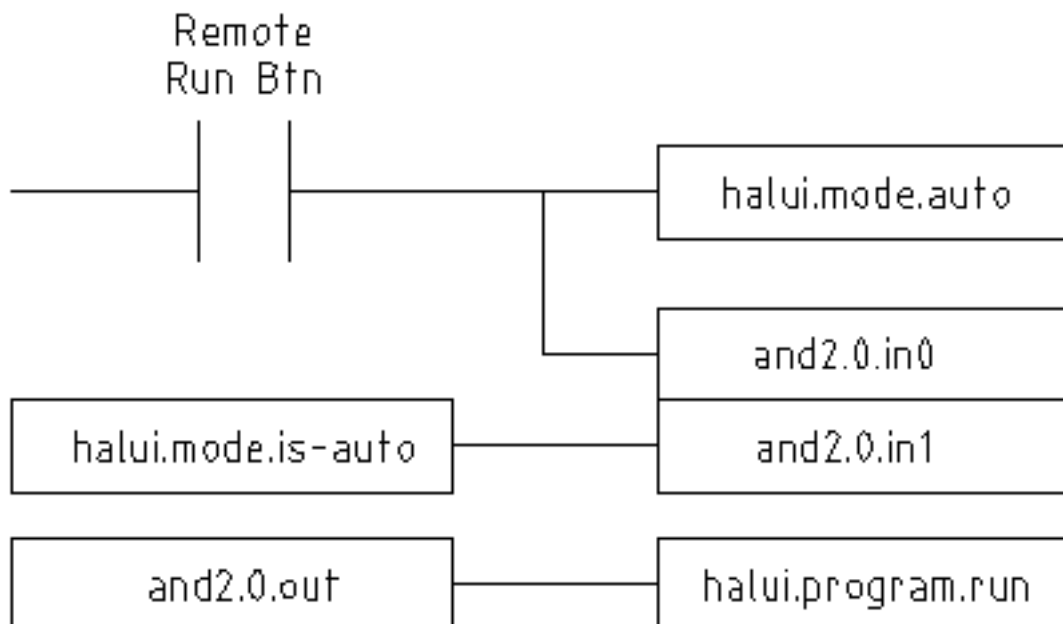


Figure 15.2: Remote Start Example (Schema)

The traditional hardware design as described above ends at the edge of the main control. Outside the control are a bunch of relatively simple boxes, connected together to do whatever is needed. Inside, the control is a big mystery — one huge black box that we hope works.

HAL extends this traditional hardware design method to the inside of the big black box. It makes device drivers and even some internal parts of the controller into smaller black boxes that can be interconnected and even replaced just like the external hardware. It allows the *system wiring diagram* to show part of the internal controller, rather than just a big black box. And most importantly, it allows the integrator to test and modify the controller using the same methods he would use on the rest of the hardware.

Terms like motors, amps, and encoders are familiar to most machine integrators. When we talk about using extra flexible eight conductor shielded cable to connect an encoder to the servo input board in the computer, the reader immediately understands what it is and is led to the question, *what kinds of connectors will I need to make up each end*. The same sort of thinking is essential for the HAL but the specific train of thought may take a bit to get on track. Using HAL words may seem a bit strange at first, but the concept of working from one connection to the next is the same.

This idea of extending the wiring diagram to the inside of the controller is what HAL is all about. If you are comfortable with the idea of interconnecting hardware black boxes, you will probably have little trouble using HAL to interconnect software black boxes.

15.1.2 HAL Concepts

This section is a glossary that defines key HAL terms but it is a bit different than a traditional glossary because these terms are not arranged in alphabetical order. They are arranged by their relationship or flow in the HAL way of things.

Component

When we talked about hardware design, we referred to the individual pieces as *parts*, *building blocks*, *black boxes*, etc. The HAL equivalent is a *component* or *HAL component*. (This document uses *HAL component* when there is likely to be confusion with other kinds of components, but

normally just uses *component*.) A HAL component is a piece of software with well-defined inputs, outputs, and behavior, that can be installed and interconnected as needed.

Note

Many HAL Components model the behaviour of a tangible part of a machine, and a **pin** may indeed be meant to be connected to a **physical pin** on the device to communicate with it, hence the names. But most often this is not the case. Imagine a retrofit of a manual lathe/mill. What LinuxCNC implements is how the machine presents itself to the outside world, and it is secondary if the implementation how to draw a circle is implemented on the machine already or provided from LinuxCNC. And it is common to add buttons to the imaginary retrofit that **signal** an action, like an emergency stop. LinuxCNC and the machine become one. And that is through the HAL.

Parameter

Many hardware components have adjustments that are not connected to any other components but still need to be accessed. For example, servo amps often have trim pots to allow for tuning adjustments, and test points where a meter or scope can be attached to view the tuning results. HAL components also can have such items, which are referred to as *parameters*. There are two types of parameters: Input parameters are equivalent to trim pots - they are values that can be adjusted by the user, and remain fixed once they are set. Output parameters cannot be adjusted by the user - they are equivalent to test points that allow internal signals to be monitored.

Pin

Hardware components have terminals which are used to interconnect them. The HAL equivalent is a *pin* or *HAL pin*. (*HAL pin* is used when needed to avoid confusion.) All HAL pins are named, and the pin names are used when interconnecting them. HAL pins are software entities that exist only inside the computer.

Physical_Pin

Many I/O devices have real physical pins or terminals that connect to external hardware, for example the pins of a parallel port connector. To avoid confusion, these are referred to as *physical pins*. These are the things that *stick out* into the real world.

Note

You may be wondering what relationship there is between the HAL_pins, Physical_pins and external elements like encoders or a STG card: we are dealing here with interfaces of data translation/conversion type.

Signal

In a physical machine, the terminals of real hardware components are interconnected by wires. The HAL equivalent of a wire is a *signal* or *HAL signal*. HAL signals connect HAL pins together as required by the machine builder. HAL signals can be disconnected and reconnected at will (even while the machine is running).

Type

When using real hardware, you would not connect a 24 volt relay output to the +/-10V analog input of a servo amp. HAL pins have the same restrictions, which are based upon their type. Both pins and signals have types, and signals can only be connected to pins of the same type. Currently there are 4 types, as follows:

- bit - a single TRUE/FALSE or ON/OFF value
 - float - a 64 bit floating point value, with approximately 53 bits of resolution and over 1000 bits of dynamic range.
-

- `u32` - a 32 bit unsigned integer, legal values are 0 to 4,294,967,295
- `s32` - a 32 bit signed integer, legal values are -2,147,483,647 to +2,147,483,647

Function

Real hardware components tend to act immediately on their inputs. For example, if the input voltage to a servo amp changes, the output also changes automatically. However software components cannot act *automatically*. Each component has specific code that must be executed to do whatever that component is supposed to do. In some cases, that code simply runs as part of the component. However in most cases, especially in realtime components, the code must run in a specific sequence and at specific intervals. For example, inputs should be read before calculations are performed on the input data, and outputs should not be written until the calculations are done. In these cases, the code is made available to the system in the form of one or more *functions*. Each function is a block of code that performs a specific action. The system integrator can use *threads* to schedule a series of functions to be executed in a particular order and at specific time intervals.

Thread

A *thread* is a list of functions that runs at specific intervals as part of a realtime task. When a thread is first created, it has a specific time interval (period), but no functions. Functions can be added to the thread, and will be executed in order every time the thread runs.

As an example, suppose we have a parport component named `hal_parport`. That component defines one or more HAL pins for each physical pin. The pins are described in that component's doc section: their names, how each pin relates to the physical pin, are they inverted, can you change polarity, etc. But that alone doesn't get the data from the HAL pins to the physical pins. It takes code to do that, and that is where functions come into the picture. The parport component needs at least two functions: one to read the physical input pins and update the HAL pins, the other to take data from the HAL pins and write it to the physical output pins. Both of these functions are part of the parport driver.

15.1.3 HAL components

Each HAL component is a piece of software with well-defined inputs, outputs, and behavior, that can be installed and interconnected as needed. The section [Realtime Components List](#) lists all available components and a brief description of what each does.

15.1.4 Timing Issues In HAL

Unlike the physical wiring models between black boxes that we have said that HAL is based upon, simply connecting two pins with a `hal-signal` falls far short of the action of the physical case.

True relay logic consists of relays connected together, and when a contact opens or closes, current flows (or stops) immediately. Other coils may change state, etc, and it all just *happens*. But in PLC style ladder logic, it doesn't work that way. Usually in a single pass through the ladder, each rung is evaluated in the order in which it appears, and only once per pass. A perfect example is a single rung ladder, with a NC contact in series with a coil. The contact and coil belong to the same relay.

If this were a conventional relay, as soon as the coil is energized, the contacts begin to open and de-energize it. That means the contacts close again, etc, etc. The relay becomes a buzzer.

With a PLC, if the coil is OFF and the contact is closed when the PLC begins to evaluate the rung, then when it finishes that pass, the coil is ON. The fact that turning on the coil opens the contact feeding it is ignored until the next pass. On the next pass, the PLC sees that the contact is open, and de-energizes the coil. So the relay still switches rapidly between on and off, but at a rate determined by how often the PLC evaluates the rung.

In HAL, the function is the code that evaluates the rung(s). In fact, the HAL-aware realtime version of ClassicLadder exports a function to do exactly that. Meanwhile, a thread is the thing that runs the

function at specific time intervals. Just like you can choose to have a PLC evaluate all its rungs every 10 ms, or every second, you can define HAL threads with different periods.

What distinguishes one thread from another is *not* what the thread does - that is determined by which functions are connected to it. The real distinction is simply how often a thread runs.

In LinuxCNC you might have a 50 μ s thread and a 1 ms thread. These would be created based on `BASE_PERIOD` and `SERVO_PERIOD`, the actual times depend on the values in your ini file.

The next step is to decide what each thread needs to do. Some of those decisions are the same in (nearly) any LinuxCNC system—For instance, motion-command-handler is always added to servo-thread.

Other connections would be made by the integrator. These might include hooking the STG driver's encoder read and DAC write functions to the servo thread, or hooking stepgen's function to the base-thread, along with the parport function(s) to write the steps to the port.

15.2 HAL Basics

This document provides a reference to the basics of HAL.

15.2.1 HAL Commands

More detailed information can be found in the man page for `halcmd`: run *man halcmd* in a terminal window.

To see the HAL configuration and check the status of pins and parameters use the HAL Configuration window on the Machine menu in AXIS. To watch a pin status open the Watch tab and click on each pin you wish to watch and it will be added to the watch window.

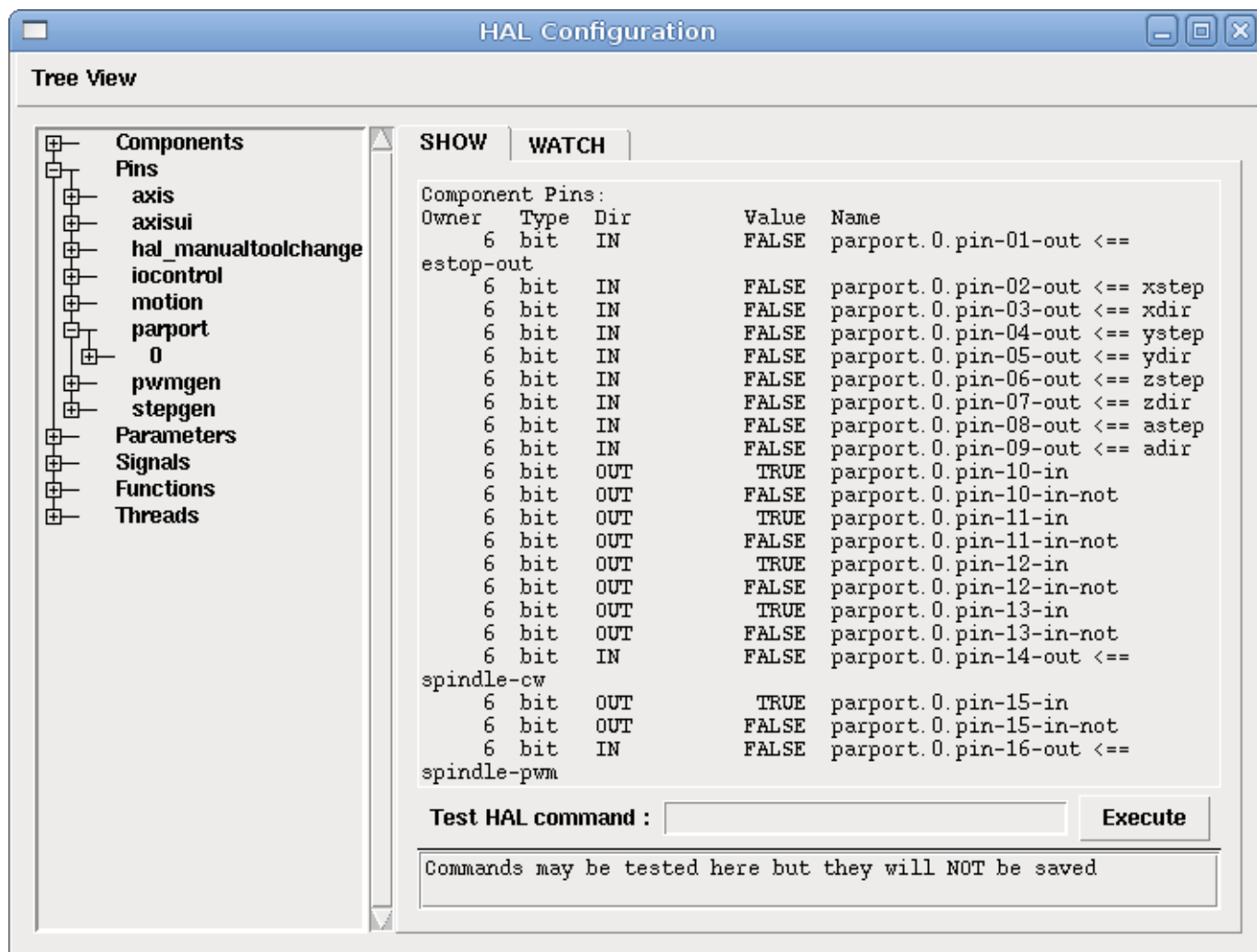


Figure 15.3: HAL Configuration Window

15.2.1.1 loadrt

The command *loadrt* loads a real time HAL component. Real time component functions need to be added to a thread to be updated at the rate of the thread. You cannot load a user space component into the real time space.

loadrt Syntax and Example

```
loadrt <component> <options>
loadrt mux4 count=1
```

15.2.1.2 addf

The *addf* command adds a function to a real-time thread. If the StepConf wizard was used to create the configuration, two threads have been created (`base-thread` and `servo-thread`).

addf adds function *funcname* to thread *threadname*. Default is to add the function in the order they are in the file. If *position* is specified, adds the function to that spot in the thread. Negative *position* means position with respect to the end of the thread. For example 1 is start of thread, -1 is the end of the thread, -3 is third from the end.

For some functions it is important to load them in a certain order, like the parport read and write functions. The function name is usually the component name plus a number. In the following example the component *or2* is loaded and *show function* shows the name of the *or2* function

```
$ halrun
halcmd: loadrt or2
halcmd: show function
Exported Functions:
Owner  CodeAddr  Arg      FP  Users  Name
00004  f8bc5000  f8f950c8  NO   0      or2.0
```

You have to add a function from a HAL real time component to a thread to get the function to update at the rate of the thread.

Usually there are two threads as shown in this example. Some components use floating point math and must be added to a thread that supports floating point math. The *FP* indicates if floating point math is supported in that thread.

```
$ halrun
halcmd: loadrt motmod base_period_nsec=55555 servo_period_nsec=1000000 num_joints=3
halcmd: show thread
Realtime Threads:
  Period  FP    Name              (    Time, Max-Time )
  995976  YES    servo-thread (    0,      0 )
  55332   NO    base-thread  (    0,      0 )
```

- base-thread (the high-speed thread): this thread handles items that need a fast response, like making step pulses, and reading and writing the parallel port. Does not support floating point math.
- servo-thread (the slow-speed thread): this thread handles items that can tolerate a slower response, like the motion controller, ClassicLadder, and the motion command handler and supports floating point math.

addf Syntax and Example

```
addf <function> <thread>
addf mux4.0 servo-thread
```

Note

If the component requires a floating point thread that is usually the slower servo-thread.

15.2.1.3 loadusr

The command *loadusr* loads a user space HAL component. User space programs are their own separate processes, which optionally talk to other HAL components via pins and parameters. You cannot load real time components into user space.

Flags may be one or more of the following:

- W to wait for the component to become ready. The component is assumed to have the same name as the first argument of the command.
 - Wn <name> to wait for the component, which will have the given <name>. This only applies if the component has a name option.
-

- w to wait for the program to exit
- i to ignore the program return value (with -w)
- n name a component when it is a valid option for that component.

loadusr Syntax and Examples

```
loadusr <component> <options>
loadusr halui
loadusr -Wn spindle gs2_vfd -n spindle
```

In English it means *loadusr wait for name spindle component gs2_vfd name spindle*.

15.2.1.4 net

The command *net* creates a *connection* between a signal and one or more pins. If the signal does not exist *net* creates the new signal. This replaces the need to use the command *newsig*. The optional direction arrows *<=*, *=>* and *<=>* make it easier to follow the logic when reading a *net* command line and are not used by the *net* command. The direction arrows must be separated by a space from the pin names.

net Syntax and Example

```
net signal-name pin-name <optional arrow> <optional second pin-name>
net home-x joint.0.home-sw-in <= parport.0.pin-11-in
```

In the above example *home-x* is the signal name, *joint.0.home-sw-in* is a *Direction IN* pin, *<=* is the optional direction arrow, and *parport.0.pin-11-in* is a *Direction OUT* pin. This may seem confusing but the in and out labels for a parallel port pin indicates the physical way the pin works not how it is handled in HAL.

A pin can be connected to a signal if it obeys the following rules:

- An IN pin can always be connected to a signal.
- An IO pin can be connected unless there's an OUT pin on the signal.
- An OUT pin can be connected only if there are no other OUT or IO pins on the signal.

The same *signal-name* can be used in multiple *net* commands to connect additional pins, as long as the rules above are obeyed.



Figure 15.4: Signal Direction

This example shows the signal `xStep` with the source being `stepgen.0.out` and with two readers, `parport.0.pin-02-out` and `parport.0.pin-08-out`. Basically the value of `stepgen.0.out` is sent to the signal `xStep` and that value is then sent to `parport.0.pin-02-out` and `parport.0.pin-08-out`.

```
# signal    source          destination    destination
net xStep stepgen.0.out => parport.0.pin-02-out parport.0.pin-08-out
```

Since the signal `xStep` contains the value of `stepgen.0.out` (the source) you can use the same signal again to send the value to another reader. To do this just use the signal with the readers on another line.

```
# signal    destination2
net xStep => parport.0.pin-06-out
```

I/O pins An I/O pin like `encoder.N.index-enable` can be read or set as allowed by the component.

15.2.1.5 setp

The command `setp` sets the value of a pin or parameter. The valid values will depend on the type of the pin or parameter. It is an error if the data types do not match.

Some components have parameters that need to be set before use. Parameters can be set before use or while running as needed. You cannot use `setp` on a pin that is connected to a signal.

setp Syntax and Example

```
setp <pin/parameter-name> <value>
setp parport.0.pin-08-out TRUE
```

15.2.1.6 sets

The command *sets* sets the value of a signal.

sets Syntax and Example:

```
sets <signal-name> <value>
net mysignal and2.0.in0 pyvcp.my-led
sets mysignal 1
```

It is an error if:

- The signal-name does not exist
- If the signal already has a writer
- If value is not the correct type for the signal

15.2.1.7 unlinkp

The command *unlinkp* unlinks a pin from the connected signal. If no signal was connected to the pin prior running the command, nothing happens. The *unlinkp* command is useful for trouble shooting.

unlinkp syntax and Example

```
unlinkp <pin-name>
unlinkp parport.0.pin-02-out
```

15.2.1.8 Obsolete Commands

The following commands are depreciated and may be removed from future versions. Any new configuration should use the [net](#) command. These commands are included so older configurations will still work.

The command *linksp* creates a *connection* between a signal and one pin.

linksp Syntax and Example

```
linksp <signal-name> <pin-name>
linksp X-step parport.0.pin-02-out
```

The *linksp* command has been superseded by the *net* command.

The command *linkps* creates a *connection* between one pin and one signal. It is the same as *linksp* but the arguments are reversed.

linkps Syntax and Example

```
linkps <pin-name> <signal-name>
linkps parport.0.pin-02-out X-Step
```

The *linkps* command has been superseded by the *net* command.

the command *newsig* creates a new HAL signal by the name <signame> and the data type of <type>. Type must be *bit*, *s32*, *u32* or *float*. Error if <signame> all ready exists.

newsig Syntax and Example

```
newsig <signame> <type>
newsig Xstep bit
```

More information can be found in the HAL manual or the man pages for `halrun`.

15.2.2 HAL Data

15.2.2.1 Bit

A bit value is an on or off.

- bit values = true or 1 and false or 0 (True, TRUE, true are all valid)

15.2.2.2 Float

A *float* is a floating point number. In other words the decimal point can move as needed.

- float values = a 64 bit floating point value, with approximately 53 bits of resolution and over 2^{10} (~ 1000) bits of dynamic range.

For more information on floating point numbers see:

http://en.wikipedia.org/wiki/Floating_point

15.2.2.3 s32

An s32 number is a whole number that can have a negative or positive value.

- s32 values = integer numbers from -2147483648 to 2147483647

15.2.2.4 u32

A u32 number is a whole number that is positive only.

- u32 values = integer numbers from 0 to 4294967295

15.2.3 HAL Files

If you used the Stepper Config Wizard to generate your config you will have up to three HAL files in your config directory.

- *my-mill.hal* (if your config is named *my-mill*) This file is loaded first and should not be changed if you used the Stepper Config Wizard.
 - *custom.hal* This file is loaded next and before the GUI loads. This is where you put your custom HAL commands that you want loaded before the GUI is loaded.
 - *custom_postgui.hal* This file is loaded after the GUI loads. This is where you put your custom HAL commands that you want loaded after the GUI is loaded. Any HAL commands that use PyVCP widgets need to be placed here.
-

15.2.4 HAL Parameter

Two parameters are automatically added to each HAL component when it is created. These parameters allow you to scope the execution time of a component.

- `.time` Time is the number of CPU cycles it took to execute the function.
- `.tmax` Tmax is the maximum number of CPU cycles it took to execute the function.

tmax is a read/write parameter so the user can set it to 0 to get rid of the first time initialization on the function's execution time.

15.2.5 Basic Logic Components

HAL contains several real time logic components. Logic components follow a *Truth Table* that states what the output is for any given input. Typically these are bit manipulators and follow electrical logic gate truth tables.

For further components see [Realtime Components List](#) or the man pages.

15.2.5.1 and2

The *and2* component is a two input *and* gate. The truth table below shows the output based on each combination of input.

Syntax

```
and2 [count=N] | [names=name1[,name2...]]
```

Functions

```
and2.n
```

Pins

```
and2.N.in0 (bit, in)
and2.N.in1 (bit, in)
and2.N.out (bit, out)
```

Table 15.1: and2 Truth Table

in0	in1	out
False	False	False
True	False	False
False	True	False
True	True	True

15.2.5.2 not

The *not* component is a bit inverter.

Syntax

```
not [count=n] | [names=name1[,name2...]]
```

Functions

```
not.all  
not.n
```

Pins

```
not.n.in (bit, in)  
not.n.out (bit, out)
```

Table 15.2: not Truth Table

in	out
True	False
False	True

15.2.5.3 or2

The *or2* component is a two input OR gate.

Syntax

```
or2[count=n] | [names=name1[,name2...]]
```

Functions

```
or2.n
```

Pins

```
or2.n.in0 (bit, in)  
or2.n.in1 (bit, in)  
or2.n.out (bit, out)
```

Table 15.3: or2 Truth Table

in0	in1	out
True	False	True
True	True	True
False	True	True
False	False	False

15.2.5.4 xor2

The *xor2* component is a two input XOR (exclusive OR) gate.

Syntax

```
xor2[count=n] | [names=name1[,name2...]]
```

Functions

```
xor2.n
```

Pins

```
xor2.n.in0 (bit, in)
xor2.n.in1 (bit, in)
xor2.n.out (bit, out)
```

Table 15.4: xor2 Truth Table

in0	in1	out
True	False	True
True	True	False
False	True	True
False	False	False

15.2.6 Logic Examples

and2 example connecting two inputs to one output

```
loadrt and2 count=1
addf and2.0 servo-thread
net my-sigin1 and2.0.in0 <= parport.0.pin-11-in
net my-sigin2 and2.0.in1 <= parport.0.pin-12-in
net both-on parport.0.pin-14-out <= and2.0.out
```

In the above example one copy of *and2* is loaded into real time space and added to the servo thread. Next pin-11 of the parallel port is connected to the *in0* bit of the and gate. Next pin-12 is connected to the *in1* bit of the and gate. Last we connect the *and2* out bit to the parallel port pin-14. So following the truth table for *and2* if pin 11 and pin 12 are on then the output pin 14 will be on.

15.2.7 Conversion Components

15.2.7.1 weighted_sum

The weighted sum converts a group of bits into an integer. The conversion is the sum of the *weights* of the bits present plus any offset. It's similar to *binary coded decimal* but with more options. The *hold* bit interrupts the input processing, so that the *sum* value no longer changes.

weighted_sum component loading syntax

```
loadrt weighted_sum wsum_sizes=size[,size,...]
```

Creates groups of `weighted_sum`'s, each with the given number of input bits (size).

To update the `weighted_sum`, the `process_wsums` must be attached to a thread.

add process_wsums function

```
addf process_wsums servo-thread
```

Which updates the `weighted_sum` component.

In the following example, a copy of the AXIS HAL configuration window, bits 0 and 2 are TRUE, they have no offset. The weight (*weight*) of bit 0 is 1, that of bit 2 is 4, so the sum is 5.

weighted_sum Example

Component Pins:

Owner	Type	Dir	Value	Name
10	bit	In	TRUE	wsum.0.bit.0.in
10	s32	I/O	1	wsum.0.bit.0.weight
10	bit	In	FALSE	wsum.0.bit.1.in
10	s32	I/O	2	wsum.0.bit.1.weight
10	bit	In	TRUE	wsum.0.bit.2.in
10	s32	I/O	4	wsum.0.bit.2.weight
10	bit	In	FALSE	wsum.0.bit.3.in
10	s32	I/O	8	wsum.0.bit.3.weight
10	bit	In	FALSE	wsum.0.hold
10	s32	I/O	0	wsum.0.offset
10	s32	Out	5	wsum.0.sum

15.3 HAL TWOPASS

15.3.1 TWOPASS

LinuxCNC 2.5 supports TWOPASS processing of LinuxCNC configuration files that can help in the modularization and readability of LinuxCNC files. (LinuxCNC files are specified in an LinuxCNC ini file in the LinuxCNC stanza as [LinuxCNC]HALFILE=filename).

Normally, a set of one or more LinuxCNC configuration files must use a single, unique `loadrt` line to load a kernel module that may handle multiple instances of a component. For example, if you use a two input AND gate component (`and2`) in three different places in your setup, you would need to have a single line somewhere to specify:

```
loadrt and2 count=3
```

resulting in components `and2.0`, `and2.1`, and `and2.2`.

Configurations are more readable if you specify with the `names=option` for components where it is supported, e.g.,:

```
loadrt and2 names=aa,ab,ac
```

resulting in components `aa`, `ab`, `ac`.

It can be a maintenance problem to keep track of the components and their names since when you add (or remove) a component, you must find and update the single `loadrt` directive applicable to the component.

TWOPASS processing is enabled by including an ini file parameter in the [HAL] section:

```
[HAL]
```

```
TWOPASS = anystring
```

Where “anystring” can be any non-null string. With this setting, you can have multiple specifications like:

```
loadrt and2 names=aa
...
loadrt and2 names=ab,ac
...
loadrt and2 names=ad
```

These commands can appear in different HALFILES. The HALFILES are processed in the order of their appearance in the ini file.

The TWOPASS option can be specified with options to add output for debugging (verbose) and to prevent deletion of temporary files (nodelete). The options are separated with commas.

Example

```
[HAL]
TWOPASS = on,verbose,nodelete
```

With TWOPASS processing, all [HAL]HALFILES are first read and multiple appearances of loadrt directives for each module are accumulated. User components (loadusr) are loaded in order but no other LinuxCNC commands are executed in the initial pass.

Note

User components should use the wait (-W) option to ensure the component is ready before other commands are executed.

After the initial pass, the realtime modules are loaded (loadrt) automatically with a number equal to the total number when using the *count=* option or with all of the individual names specified when using the *names=* option.

A second pass is then made to execute all of the other LinuxCNC instructions specified in the HALFILES. The addf commands that associate a component’s functions with thread execution are executed in the order of appearance with other commands during this second pass.

While you can use either the *count=* or *names=* options, they are mutually exclusive — only one type can be specified for a given module.

TWOPASS processing is most effective when using the *names=* option. This option allows you to provide unique names that are mnemonic or otherwise relevant to the configuration. For example, if you use a derivative component to estimate the velocities and accelerations on each (x,y,z) coordinate, using the *count=* method will give arcane component names like ddt.0, ddt.1, ddt.2, etc.

Alternatively, using the *names=* option like:

```
loadrt ddt names=xvel,yvel,zvel
...
loadrt ddt names=xaccel,yaccel,zaccel
```

results in components sensibly named xvel, yvel, zvel, xaccel, yaccel, zaccel.

Many comps supplied with the distribution are created with the halcompile utility and support the *names=* option. These include the common logic components that are the glue of many LinuxCNC configurations.

User-created comps that use the halcompile utility automatically support the *names=* option as well. In addition to comps generated with the halcompile utility, numerous other comps support the *names=* option. Comps that support *names=* option include: at_pid, encoder, encoder_ratio, pid, siggen, and sim_encoder.

Two-step processing occurs before the GUI is loaded. When using a [HAL]POSTGUI_HALFILE, it is convenient to place all the [HAL]POSTGUI_HALFILE loadrt declarations for the necessary components in a preloaded halfile.

Example of a HAL section when using a POSTGUI_HALFILE

```
[HAL]
TWOPASS = on
HALFILE = core_sim.hal
HALFILE = sim_spindle_encoder.hal
HALFILE = axis_manualtoolchange.hal
HALFILE = simulated_home.hal
HALFILE = load_for_postgui.hal <- loadrt lines for components in postgui.hal

POSTGUI_HALFILE = postgui.hal
HALUI = halui
```

15.3.2 Post GUI

Some GUIs support halfiles that are processed after the GUI is started in order to connect LinuxCNC pins that are created by the GUI. When using a postgui halfile with TWOPASS processing, include all loadrt items for components added by postgui halfiles in a separate halfile that is processed before the GUI. The addf commands can also be included in the file.

Example

```
[HAL]
TWOPASS = on
HALFILE = file_1.hal
...
HALFILE = file_n.hal
HALFILE = file_with_all_loads_for_postgui.hal
...
POSTGUI_HALFILE = the_postgui_file.hal
```

15.3.3 Excluding .hal files

TWOPASS processing converts *.hal* files to equivalent *.tcl* files and uses haltcl to find loadrt and addf commands in order to accumulate and consolidate their usage. Loadrt parameters that conform to the simple *names=* (or *count=*) parameters accepted by the HAL Component Generator (*halcompile*) are expected. More complex parameter items included in specialized LinuxCNC components may not be handled properly.

A *.hal* file may be excluded from TWOPASS processing by including a magic comment line anywhere in the *.hal* file. The magic comment line must begin with the string: *#NOTWOPASS*. Files specified with this magic comment are sourced by halcmd using the *-k* (keep going if failure) and *-v* (verbose) options.

This exclusion provision can be used to isolate problems or for loading any special LinuxCNC component that does not require or benefit from TWOPASS processing.

Ordinarily, the loadrt ordering of realtime components is not critical, but loadrt ordering for special components can be enforced by placing the such loadrt directives in an excluded file.

Note

While the order of loadrt directives is not usually critical, ordering of addf directives is often very important for proper operation of servo loop components.

Excluded .hal file example

```
$ cat twopass_excluded.hal
# The following magic comment causes this file to
# be excluded from twopass processing:
# NOTWOPASS

# debugging component with complex options:
loadrt mycomponent parm1="abc def" parm2=ghi
show pin mycomponent

# ordering special components
loadrt component_1
loadrt component_2
```

Note

Case and whitespace within the magic comment are ignored. The loading of components that use *names=* or *count=* parameters (typically built by halcompile) should not be used in excluded files as that would eliminate the benefits of TWOPASS processing. The LinuxCNC commands that create signals (net) and commands that establish execution order (addf) should not be placed in excluded files. This is especially true for addf commands since their ordering may be important.

15.3.4 Examples

Examples of TWOPASS usage for a simulator are included in the directories:

```
configs/sim/axis/twopass/
configs/sim/axis/simtcl/
```

15.4 HAL Tutorial

15.4.1 Introduction

Configuration moves from theory to device — HAL device that is. For those who have had just a bit of computer programming, this section is the *Hello World* of the HAL.

Halrun can be used to create a working system. It is a command line or text file tool for configuration and tuning.

The following examples illustrate its setup and operation.

15.4.2 Halcmd

halcmd is a command line tool for manipulating HAL. A more complete man page exists for halcmd and installed together with LinuxCNC, from source or from a package. If LinuxCNC has been compiled as *run-in-place*, the man page is not installed but is accessible in the LinuxCNC main directory with the following command:

```
$ man -M docs/man halcmd
```

15.4.2.1 Notation

Terminal commands are shown without the system prompt unless you are running *HAL*. The terminal window is in *Applications/Accessories* from the main Ubuntu menu bar.

Terminal Command Example

```
me@computer:~linuxcnc$ halrun
(will be shown like the following line)
halrun

(the halcmd: prompt will be shown when running HAL)
halcmd: loadrt counter
halcmd: show pin
```

15.4.2.2 Tab-completion

Your version of `halcmd` may include tab-completion. Instead of completing file names as a shell does, it completes commands with HAL identifiers. You will have to type enough letters for a unique match. Try pressing `tab` after starting a HAL command:

Tab-completion

```
halcmd: loa<TAB>
halcmd: load
halcmd: loadrt
halcmd: loadrt cou<TAB>
halcmd: loadrt counter
```

15.4.2.3 The RTAPI environment

RTAPI stands for Real Time Application Programming Interface. Many HAL components work in realtime, and all HAL components store data in shared memory so realtime components can access it. Normal Linux does not support realtime programming or the type of shared memory that HAL needs. Fortunately there are realtime operating systems (RTOS's) that provide the necessary extensions to Linux. Unfortunately, each RTOS does things a little differently.

To address these differences, the LinuxCNC team came up with RTAPI, which provides a consistent way for programs to talk to the RTOS. If you are a programmer who wants to work on the internals of LinuxCNC, you may want to study `linuxcnc/src/rtapi/rtapi.h` to understand the API. But if you are a normal person all you need to know about RTAPI is that it (and the RTOS) needs to be loaded into the memory of your computer before you do anything with HAL.

15.4.3 A Simple Example

15.4.3.1 Loading a component

For this tutorial, we are going to assume that you have successfully installed the Live CD and, if using a RIP¹, invoke the `rip-environment` script to prepare your shell. In that case, all you need to do is load the required RTOS and RTAPI modules into memory. Just run the following command from a terminal window: **Loading HAL**

```
cd linuxcnc
halrun
halcmd:
```

¹Run In Place, when the source files have been downloaded to a user directory.

With the realtime OS and RTAPI loaded, we can move into the first example. Notice that the prompt is now shown as *halcmd*:. This is because subsequent commands will be interpreted as HAL commands, not shell commands.

For the first example, we will use a HAL component called *siggen*, which is a simple signal generator. A complete description of the *siggen* component can be found in the [Siggen](#) section of this Manual. It is a realtime component, implemented as a Linux kernel module. To load *siggen* use the HAL command *loadrt*.

Loading siggen

```
halcmd: loadrt siggen
```

15.4.3.2 Examining the HAL

Now that the module is loaded, it is time to introduce *halcmd*, the command line tool used to configure the HAL. This tutorial will introduce some *halcmd* features, for a more complete description try *man halcmd*, or see the reference in [Hal Commands](#) section of this document. The first *halcmd* feature is the *show* command. This command displays information about the current state of the HAL. To show all installed components:

Show Components

```
halcmd: show comp
```

Loaded HAL Components:				
ID	Type	Name	PID	State
3	RT	siggen		ready
2	User	halcmd2177	2177	ready

Since *halcmd* itself is a HAL component, it will always show up in the list. The number after *halcmd* in the component list is the process ID. It is possible to run more than one copy of *halcmd* at the same time (in different windows for example), so the PID is added to the end of the name to make it unique. The list also shows the *siggen* component that we installed in the previous step. The *RT* under *Type* indicates that *siggen* is a realtime component. The *User* under *Type* indicates it is a user space component.

Next, let's see what pins *siggen* makes available:

Show Pins

```
halcmd: show pin
```

Component Pins:				
Owner	Type	Dir	Value	Name
3	float	IN	1	siggen.0.amplitude
3	bit	OUT	FALSE	siggen.0.clock
3	float	OUT	0	siggen.0.cosine
3	float	IN	1	siggen.0.frequency
3	float	IN	0	siggen.0.offset
3	float	OUT	0	siggen.0.sawtooth
3	float	OUT	0	siggen.0.sine
3	float	OUT	0	siggen.0.square
3	float	OUT	0	siggen.0.triangle

This command displays all of the pins in the current HAL. A complex system could have dozens or hundreds of pins. But right now there are only nine pins. Of these pins eight are floating point and one is bit (boolean). Six carry data out of the *siggen* component and three are used to transfer settings into the component. Since we have not yet executed the code contained within the component, some the pins have a value of zero.

The next step is to look at parameters:

Show Parameters

```
halcmd: show param
```

Parameters:

Owner	Type	Dir	Value	Name
3	s32	R0	0	siggen.0.update.time
3	s32	RW	0	siggen.0.update.tmax

The *show param* command shows all the parameters in the HAL. Right now each parameter has the default value it was given when the component was loaded. Note the column labeled *Dir*. The parameters labeled *-W* are writable ones that are never changed by the component itself, instead they are meant to be changed by the user to control the component. We will see how to do this later. Parameters labeled *R-* are read only parameters. They can be changed only by the component. Finally, parameter labeled *RW* are read-write parameters. That means that they are changed by the component, but can also be changed by the user. Note: the parameters *siggen.0.update.time* and *siggen.0.update.tmax* are for debugging purposes, and won't be covered in this section.

Most realtime components export one or more functions to actually run the realtime code they contain. Let's see what function(s) *siggen* exported:

Show Functions

```
halcmd: show funct
```

Exported Functions:

Owner	CodeAddr	Arg	FP	Users	Name
00003	f801b000	fae820b8	YES	0	siggen.0.update

The *siggen* component exported a single function. It requires floating point. It is not currently linked to any threads, so *users* is zero. ²

15.4.3.3 Making realtime code run

To actually run the code contained in the function *siggen.0.update*, we need a realtime thread. The component called *threads* that is used to create a new thread. Lets create a thread called *test-thread* with a period of 1 ms (1,000 µs or 1,000,000 ns):

```
halcmd: loadrt threads name1=test-thread period1=1000000
```

Let's see if that worked:

Show Threads

```
halcmd: show thread
```

Realtime Threads:

Period	FP	Name	(Time, Max-Time)
999855	YES	test-thread	(0, 0)

It did. The period is not exactly 1,000,000 ns because of hardware limitations, but we have a thread that runs at approximately the correct rate, and which can handle floating point functions. The next step is to connect the function to the thread:

Add Function

```
halcmd: addf siggen.0.update test-thread
```

²CodeAddr and Arg fields were used during development and should probably disappear.

Up till now, we've been using *halcmd* only to look at the HAL. However, this time we used the *addf* (add function) command to actually change something in the HAL. We told *halcmd* to add the function *siggen.0.update* to the thread *test-thread*, and if we look at the thread list again, we see that it succeeded:

```
halcmd: show thread
```

```
Realtime Threads:
  Period  FP      Name              (      Time, Max-Time )
  999855   YES    test-thread          (          0,          0 )
                        1 siggen.0.update
```

There is one more step needed before the *siggen* component starts generating signals. When the HAL is first started, the thread(s) are not actually running. This is to allow you to completely configure the system before the realtime code starts. Once you are happy with the configuration, you can start the realtime code like this:

```
halcmd: start
```

Now the signal generator is running. Let's look at its output pins:

```
halcmd: show pin
```

```
Component Pins:
Owner  Type  Dir      Value  Name
  3  float IN          1  siggen.0.amplitude
  3   bit  OUT        FALSE  siggen.0.clock
  3  float OUT   -0.1640929  siggen.0.cosine
  3  float IN          1  siggen.0.frequency
  3  float IN          0  siggen.0.offset
  3  float OUT   -0.4475303  siggen.0.sawtooth
  3  float OUT    0.9864449  siggen.0.sine
  3  float OUT         -1  siggen.0.square
  3  float OUT   -0.1049393  siggen.0.triangle
```

And let's look again:

```
halcmd: show pin
```

```
Component Pins:
Owner  Type  Dir      Value  Name
  3  float IN          1  siggen.0.amplitude
  3   bit  OUT        FALSE  siggen.0.clock
  3  float OUT    0.0507619  siggen.0.cosine
  3  float IN          1  siggen.0.frequency
  3  float IN          0  siggen.0.offset
  3  float OUT   -0.516165  siggen.0.sawtooth
  3  float OUT    0.9987108  siggen.0.sine
  3  float OUT         -1  siggen.0.square
  3  float OUT    0.03232994  siggen.0.triangle
```

We did two *show pin* commands in quick succession, and you can see that the outputs are no longer zero. The sine, cosine, sawtooth, and triangle outputs are changing constantly. The square output is also working, however it simply switches from +1.0 to -1.0 every cycle.

15.4.3.4 Changing Parameters

The real power of HAL is that you can change things. For example, we can use the *setp* command to set the value of a parameter. Let's change the amplitude of the signal generator from 1.0 to 5.0:

Set Pin

```
halcmd: setp siggen.0.amplitude 5
```

Check the parameters and pins again

```
halcmd: show param
```

Parameters:

Owner	Type	Dir	Value	Name
3	s32	R0	1754	siggen.0.update.time
3	s32	RW	16997	siggen.0.update.tmax

```
halcmd: show pin
```

Component Pins:

Owner	Type	Dir	Value	Name
3	float	IN	5	siggen.0.amplitude
3	bit	OUT	FALSE	siggen.0.clock
3	float	OUT	0.8515425	siggen.0.cosine
3	float	IN	1	siggen.0.frequency
3	float	IN	0	siggen.0.offset
3	float	OUT	2.772382	siggen.0.sawtooth
3	float	OUT	-4.926954	siggen.0.sine
3	float	OUT	5	siggen.0.square
3	float	OUT	0.544764	siggen.0.triangle

Note that the value of parameter *siggen.0.amplitude* has changed to 5, and that the pins now have larger values.

15.4.3.5 Saving the HAL configuration

Most of what we have done with *halcmd* so far has simply been viewing things with the *show* command. However two of the commands actually changed things. As we design more complex systems with HAL, we will use many commands to configure things just the way we want them. HAL has the memory of an elephant, and will retain that configuration until we shut it down. But what about next time? We don't want to manually enter a bunch of commands every time we want to use the system. We can save the configuration of the entire HAL with a single command:

Save

```
halcmd: save
```

```
# components
loadrt threads name1=test-thread period1=1000000
loadrt siggen
# pin aliases
# signals
# nets
# parameter values
setp siggen.0.update.tmax 14687
# realtime thread/function links
addf siggen.0.update test-thread
```

The output of the *save* command is a sequence of HAL commands. If you start with an *empty* HAL and run all these commands, you will get the configuration that existed when the *save* command was issued. To save these commands for later use, we simply redirect the output to a file:

Save to a file

```
halcmd: save all saved.hal
```

15.4.3.6 Exiting halrun

When you're finished with your HAL session type *exit* at the *halcmd:* prompt. This will return you to the system prompt and close down the HAL session. Do not simply close the terminal window without shutting down the HAL session.

Exit HAL

```
halcmd: exit
```

15.4.3.7 Restoring the HAL configuration

To restore the HAL configuration stored in *saved.hal*, we need to execute all of those HAL commands. To do that, we use *-f <file name>* which reads commands from a file, and *-I* (upper case i) which shows the halcmd prompt after executing the commands:

Run a Saved File

```
halrun -I -f saved.hal
```

Notice that there is not a *start* command in *saved.hal*. It's necessary to issue it again (or edit *saved.hal* to add it there).

15.4.3.8 Removing HAL from memory

If an unexpected shut down of a HAL session occurs you might have to unload HAL before another session can begin. To do this type the following command in a terminal window.

Removing HAL

```
halrun -U
```

15.4.4 Halmeter

You can build very complex HAL systems without ever using a graphical interface. However there is something satisfying about seeing the result of your work. The first and simplest GUI tool for the HAL is halmeter. It is a very simple program that is the HAL equivalent of the handy multimeter (or analog meter for the old timers).

It allows to observe the pins, signals or parameters by displaying the current value of these entities. It is very easy to use application for graphical environments. In a console type:

```
halmeter
```

Two windows will appear. The selection window is the largest and includes three tabs:

- One lists all the pins currently defined in HAL,
- One lists all the signals,
- One lists all the parameters.

Click on a tab, then click on one of the items to select it. The small window will show the name and value of the selected item. The display is updated approximately 10 times per second. To free screen space, the selection window can be closed with the *Close* button. On the little window, hidden under

the selection window at program launch, the *Select* button, re-opens the selection window and the *Exit* button stops the program and closes both windows.

It is possible to run several halmeters simultaneously, which makes it possible to visualize several items at the same time. To open a halmeter and release the console by running it in the background, run the following command:

```
halmeter &
```

It is possible to launch halmeter and make it immediately display an item. For this, add *pin|sig|par[am] name* arguments on the command line. It will display the signal, pin, or parameter *name* as soon as it will start. If the indicated item does not exist, it will start normally. Finally, if an item is specified for display, it is possible add *-s* in front of *pin|sig|param* to tell halmeter to use an even smaller window. The item name will be displayed in the title bar instead of below the value and there will be no button. This is useful for displaying a lot of halmeters in a small space. We will use the siggen component again to check out halmeter. If you just finished the previous example, then you can load siggen using the saved file. If not, we can load it just like we did before:

```
halrun
halcmd: loadrt siggen
halcmd: loadrt threads name1=test-thread period1=1000000
halcmd: addf siggen.0.update test-thread
halcmd: start
halcmd: setp siggen.0.amplitude 5
```

At this point we have the siggen component loaded and running. It's time to start halmeter.

Starting Halmeter

```
halcmd: loadusr halmeter
```

The first window you will see is the *Select Item to Probe* window.



Figure 15.5: Halmeter Select Window

This dialog has three tabs. The first tab displays all of the HAL pins in the system. The second one displays all the signals, and the third displays all the parameters. We would like to look at the pin *siggen.0.cosine* first, so click on it then click the *Close* button. The probe selection dialog will close, and the meter looks something like the following figure.



Figure 15.6: Halmeter Window

To change what the meter displays press the *Select* button which brings back the *Select Item to Probe* window.

You should see the value changing as siggen generates its cosine wave. Halmeter refreshes its display about 5 times per second.

To shut down halmeter, just click the exit button.

If you want to look at more than one pin, signal, or parameter at a time, you can just start more halmeters. The halmeter window was intentionally made very small so you could have a lot of them on the screen at once.

15.4.5 Stepgen Example

Up till now we have only loaded one HAL component. But the whole idea behind the HAL is to allow you to load and connect a number of simple components to make up a complex system. The next example will use two components.

Before we can begin building this new example, we want to start with a clean slate. If you just finished one of the previous examples, we need to remove the all components and reload the RTAPI and HAL libraries.

```
halcmd: exit
```

15.4.5.1 Installing the components

Now we are going to load the step pulse generator component. For a detailed description of this component refer to the stepgen section of the Integrator Manual. In this example we will use the *velocity* control type of stepgen. For now, we can skip the details, and just run the following commands.

In this example we will use the *velocity* control type from the stepgen component.

```
halrun
halcmd: loadrt stepgen step_type=0,0 ctrl_type=v,v
halcmd: loadrt siggen
halcmd: loadrt threads name1=fast fp1=0 period1=50000 name2=slow period2=1000000
```

The first command loads two step generators, both configured to generate stepping type 0. The second command loads our old friend siggen, and the third one creates two threads, a fast one with a period of 50 microseconds and a slow one with a period of 1 millisecond. The fast thread doesn't support floating point functions.

As before, we can use *halcmd show* to take a look at the HAL. This time we have a lot more pins and parameters than before:

```
halcmd: show pin
```

```
Component Pins:
Owner  Type  Dir      Value  Name
  4  float IN        1  siggen.0.amplitude
  4  bit   OUT      FALSE  siggen.0.clock
  4  float OUT        0  siggen.0.cosine
  4  float IN        1  siggen.0.frequency
  4  float IN        0  siggen.0.offset
  4  float OUT        0  siggen.0.sawtooth
  4  float OUT        0  siggen.0.sine
  4  float OUT        0  siggen.0.square
  4  float OUT        0  siggen.0.triangle
  3  s32   OUT        0  stepgen.0.counts
  3  bit   OUT      FALSE  stepgen.0.dir
```

```

3 bit IN FALSE stepgen.0.enable
3 float OUT 0 stepgen.0.position-fb
3 bit OUT FALSE stepgen.0.step
3 float IN 0 stepgen.0.velocity-cmd
3 s32 OUT 0 stepgen.1.counts
3 bit OUT FALSE stepgen.1.dir
3 bit IN FALSE stepgen.1.enable
3 float OUT 0 stepgen.1.position-fb
3 bit OUT FALSE stepgen.1.step
3 float IN 0 stepgen.1.velocity-cmd

```

```
halcmd: show param
```

Parameters:

Owner	Type	Dir	Value	Name
4	s32	R0	0	siggen.0.update.time
4	s32	RW	0	siggen.0.update.tmax
3	u32	RW	0x00000001	stepgen.0.dirhold
3	u32	RW	0x00000001	stepgen.0.dirsetup
3	float	R0	0	stepgen.0.frequency
3	float	RW	0	stepgen.0.maxaccel
3	float	RW	0	stepgen.0.maxvel
3	float	RW	1	stepgen.0.position-scale
3	s32	R0	0	stepgen.0.rawcounts
3	u32	RW	0x00000001	stepgen.0.steplen
3	u32	RW	0x00000001	stepgen.0.stepspace
3	u32	RW	0x00000001	stepgen.1.dirhold
3	u32	RW	0x00000001	stepgen.1.dirsetup
3	float	R0	0	stepgen.1.frequency
3	float	RW	0	stepgen.1.maxaccel
3	float	RW	0	stepgen.1.maxvel
3	float	RW	1	stepgen.1.position-scale
3	s32	R0	0	stepgen.1.rawcounts
3	u32	RW	0x00000001	stepgen.1.steplen
3	u32	RW	0x00000001	stepgen.1.stepspace
3	s32	R0	0	stepgen.capture-position.time
3	s32	RW	0	stepgen.capture-position.tmax
3	s32	R0	0	stepgen.make-pulses.time
3	s32	RW	0	stepgen.make-pulses.tmax
3	s32	R0	0	stepgen.update-freq.time
3	s32	RW	0	stepgen.update-freq.tmax

15.4.5.2 Connecting pins with signals

What we have is two step pulse generators, and a signal generator. Now it is time to create some HAL signals to connect the two components. We are going to pretend that the two step pulse generators are driving the X and Y axis of a machine. We want to move the table in circles. To do this, we will send a cosine signal to the X axis, and a sine signal to the Y axis. The siggen module creates the sine and cosine, but we need *wires* to connect the modules together. In the HAL, *wires* are called signals. We need to create two of them. We can call them anything we want, for this example they will be *X-vel* and *Y-vel*. The signal *X-vel* is intended to run from the cosine output of the signal generator to the velocity input of the first step pulse generator. The first step is to connect the signal to the signal generator output. To connect a signal to a pin we use the net command.

net command

```
halcmd: net X-vel <= siggen.0.cosine
```

To see the effect of the *net* command, we show the signals again.

```
halcmd: show sig
```

```
Signals:
Type      Value  Name      (linked to)
float      0    X-vel    <== siggen.0.cosine
```

When a signal is connected to one or more pins, the `show` command lists the pins immediately following the signal name. The *arrow* shows the direction of data flow - in this case, data flows from pin `siggen.0.cosine` to signal `X-vel`. Now let's connect the `X-vel` to the velocity input of a step pulse generator.

```
halcmd: net X-vel => stepgen.0.velocity-cmd
```

We can also connect up the Y axis signal `Y-vel`. It is intended to run from the sine output of the signal generator to the input of the second step pulse generator. The following command accomplishes in one line what two *net* commands accomplished for `X-vel`.

```
halcmd: net Y-vel siggen.0.sine => stepgen.1.velocity-cmd
```

Now let's take a final look at the signals and the pins connected to them.

```
halcmd: show sig
```

```
Signals:
Type      Value  Name      (linked to)
float      0    X-vel    <== siggen.0.cosine
           ==> stepgen.0.velocity-cmd
float      0    Y-vel    <== siggen.0.sine
           ==> stepgen.1.velocity-cmd
```

The *show sig* command makes it clear exactly how data flows through the HAL. For example, the `X-vel` signal comes from pin `siggen.0.cosine`, and goes to pin `stepgen.0.velocity-cmd`.

15.4.5.3 Setting up realtime execution - threads and functions

Thinking about data flowing through *wires* makes pins and signals fairly easy to understand. Threads and functions are a little more difficult. Functions contain the computer instructions that actually get things done. Thread are the method used to make those instructions run when they are needed. First let's look at the functions available to us.

```
halcmd: show funct
```

```
Exported Functions:
Owner  CodeAddr  Arg      FP  Users  Name
00004  f9992000  fc731278 YES   0    siggen.0.update
00003  f998b20f  fc7310b8 YES   0    stepgen.capture-position
00003  f998b000  fc7310b8 NO    0    stepgen.make-pulses
00003  f998b307  fc7310b8 YES   0    stepgen.update-freq
```

In general, you will have to refer to the documentation for each component to see what its functions do. In this case, the function `siggen.0.update` is used to update the outputs of the signal generator. Every time it is executed, it calculates the values of the sine, cosine, triangle, and square outputs. To make smooth signals, it needs to run at specific intervals.

The other three functions are related to the step pulse generators.

The first one, `stepgen.capture_position`, is used for position feedback. It captures the value of an internal counter that counts the step pulses as they are generated. Assuming no missed steps, this counter indicates the position of the motor.

The main function for the step pulse generator is *stepgen.make_pulses*. Every time *make_pulses* runs it decides if it is time to take a step, and if so sets the outputs accordingly. For smooth step pulses, it should run as frequently as possible. Because it needs to run so fast, *make_pulses* is highly optimized and performs only a few calculations. Unlike the others, it does not need floating point math.

The last function, *stepgen.update-freq*, is responsible for doing scaling and some other calculations that need to be performed only when the frequency command changes.

What this means for our example is that we want to run *siggen.0.update* at a moderate rate to calculate the sine and cosine values. Immediately after we run *siggen.0.update*, we want to run *stepgen.update_freq* to load the new values into the step pulse generator. Finally we need to run *stepgen.make_pulses* as fast as possible for smooth pulses. Because we don't use position feedback, we don't need to run *stepgen.capture_position* at all.

We run functions by adding them to threads. Each thread runs at a specific rate. Let's see what threads we have available.

```
halcmd: show thread
```

```
Realtime Threads:
  Period  FP      Name              (      Time, Max-Time )
  996980  YES      slow (      0,      0 )
  49849   NO      fast (      0,      0 )
```

The two threads were created when we loaded *threads*. The first one, *slow*, runs every millisecond, and is capable of running floating point functions. We will use it for *siggen.0.update* and *stepgen.update_freq*. The second thread is *fast*, which runs every 50 microseconds, and does not support floating point. We will use it for *stepgen.make_pulses*. To connect the functions to the proper thread, we use the *addf* command. We specify the function first, followed by the thread.

```
halcmd: addf siggen.0.update slow
halcmd: addf stepgen.update-freq slow
halcmd: addf stepgen.make-pulses fast
```

After we give these commands, we can run the *show thread* command again to see what happened.

```
halcmd: show thread
```

```
Realtime Threads:
  Period  FP      Name              (      Time, Max-Time )
  996980  YES      slow (      0,      0 )
                   1 siggen.0.update
                   2 stepgen.update-freq
  49849   NO      fast (      0,      0 )
                   1 stepgen.make-pulses
```

Now each thread is followed by the names of the functions, in the order in which the functions will run.

15.4.5.4 Setting parameters

We are almost ready to start our HAL system. However we still need to adjust a few parameters. By default, the *siggen* component generates signals that swing from +1 to -1. For our example that is fine, we want the table speed to vary from +1 to -1 inches per second. However the scaling of the step pulse generator isn't quite right. By default, it generates an output frequency of 1 step per second with an input of 1.000. It is unlikely that one step per second will give us one inch per second of table movement. Let's assume instead that we have a 5 turn per inch leadscrew, connected to a 200 step per rev stepper with 10x microstepping. So it takes 2000 steps for one revolution of the screw, and 5 revolutions to travel one inch. that means the overall scaling is 10000 steps per inch. We need to multiply the velocity input to the step pulse generator by 10000 to get the proper output. That is

exactly what the parameter *stepgen.n.velocity-scale* is for. In this case, both the X and Y axis have the same scaling, so we set the scaling parameters for both to 10000.

```
halcmd: setp stepgen.0.position-scale 10000
halcmd: setp stepgen.1.position-scale 10000
halcmd: setp stepgen.0.enable 1
halcmd: setp stepgen.1.enable 1
```

This velocity scaling means that when the pin *stepgen.0.velocity-cmd* is 1.000, the step generator will generate 10000 pulses per second (10 kHz). With the motor and leadscrew described above, that will result in the axis moving at exactly 1.000 inches per second. This illustrates a key HAL concept - things like scaling are done at the lowest possible level, in this case in the step pulse generator. The internal signal *X-vel* is the velocity of the table in inches per second, and other components such as *siggen* don't know (or care) about the scaling at all. If we changed the leadscrew, or motor, we would change only the scaling parameter of the step pulse generator.

15.4.5.5 Run it!

We now have everything configured and are ready to start it up. Just like in the first example, we use the *start* command.

```
halcmd: start
```

Although nothing appears to happen, inside the computer the step pulse generator is cranking out step pulses, varying from 10 kHz forward to 10 kHz reverse and back again every second. Later in this tutorial we'll see how to bring those internal signals out to run motors in the real world, but first we want to look at them and see what is happening.

15.4.6 Halscope

The previous example generates some very interesting signals. But much of what happens is far too fast to see with halmeter. To take a closer look at what is going on inside the HAL, we want an oscilloscope. Fortunately HAL has one, called halscope.

Halscope has two parts - a realtime part that is loaded as a kernel module, and a user part that supplies the GUI and display. However, you don't need to worry about this, because the userspace portion will automatically request that the realtime part be loaded. With LinuxCNC running in a terminal you can start halscope with the following command.

Starting Halscope

```
halcmd loadusr halscope
```

If LinuxCNC is not running or the autosave.halscope file does not match the pins available in the current running LinuxCNC the scope GUI window will open, immediately followed by a *Realtime function not linked* dialog that looks like the following figure. To change the sample rate left click on the samples box.



Figure 15.7: Realtime function not linked dialog

This dialog is where you set the sampling rate for the oscilloscope. For now we want to sample once per millisecond, so click on the 989 μ s thread *slow* and leave the multiplier at 1. We will also leave the record length at 4000 samples, so that we can use up to four channels at one time. When you

select a thread and then click *OK*, the dialog disappears, and the scope window looks something like the following figure.

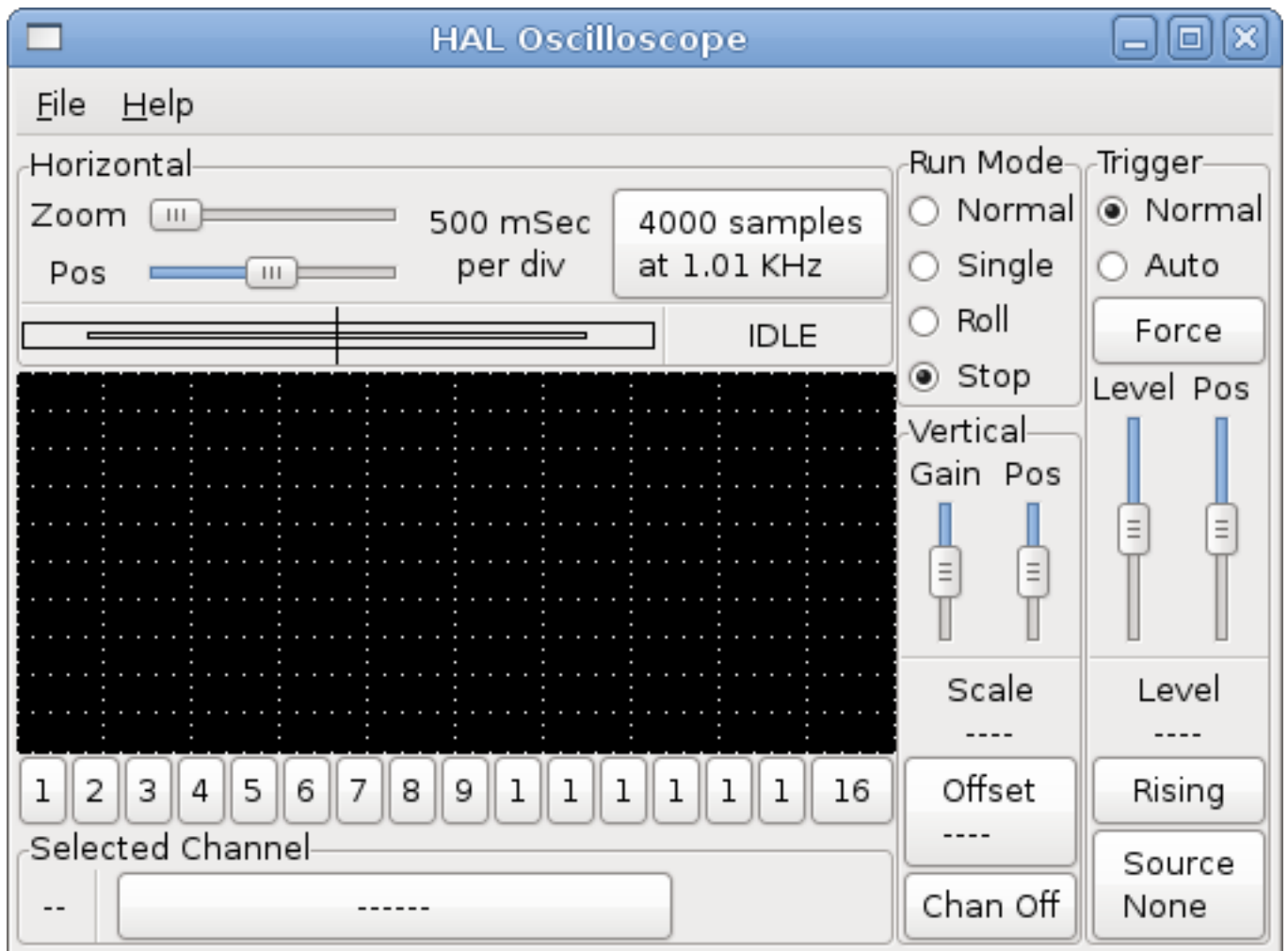


Figure 15.8: Initial scope window

15.4.6.1 Hooking up the scope probes

At this point, Halscope is ready to use. We have already selected a sample rate and record length, so the next step is to decide what to look at. This is equivalent to hooking *virtual scope probes* to the HAL. Halscope has 16 channels, but the number you can use at any one time depends on the record length - more channels means shorter records, since the memory available for the record is fixed at approximately 16,000 samples.

The channel buttons run across the bottom of the halscope screen. Click button *1*, and you will see the *Select Channel Source* dialog as shown in the following figure. This dialog is very similar to the one used by Halmeter. We would like to look at the signals we defined earlier, so we click on the *Signals* tab, and the dialog displays all of the signals in the HAL (only two for this example).



Figure 15.9: Select Channel Source

To choose a signal, just click on it. In this case, we want channel 1 to display the signal *X-vel*. Click on the Signals tab then click on *X-vel* and the dialog closes and the channel is now selected.



Figure 15.10: Select Signal

The channel 1 button is pressed in, and channel number 1 and the name *X-vel* appear below the row of buttons. That display always indicates the selected channel - you can have many channels on the screen, but the selected one is highlighted, and the various controls like vertical position and scale always work on the selected one.

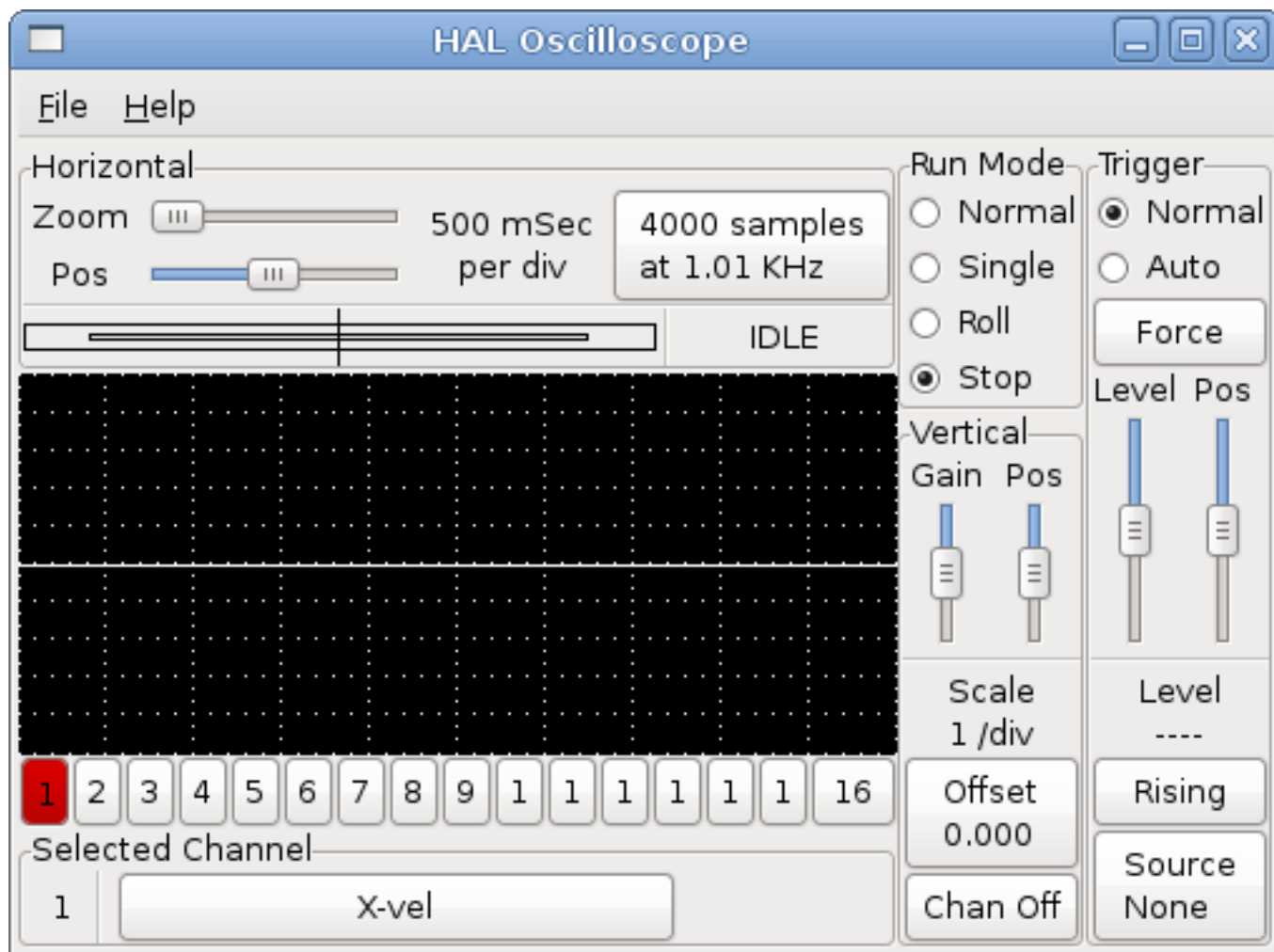


Figure 15.11: Halscope

To add a signal to channel 2, click the 2 button. When the dialog pops up, click the *Signals* tab, then click on *Y-vel*. We also want to look at the square and triangle wave outputs. There are no signals connected to those pins, so we use the *Pins* tab instead. For channel 3, select *siggen.0.triangle* and for channel 4, select *siggen.0.square*.

15.4.6.2 Capturing our first waveforms

Now that we have several probes hooked to the HAL, it's time to capture some waveforms. To start the scope, click the *Normal* button in the *Run Mode* section of the screen (upper right). Since we have a 4000 sample record length, and are acquiring 1000 samples per second, it will take halscope about 2 seconds to fill half of its buffer. During that time a progress bar just above the main screen will show the buffer filling. Once the buffer is half full, the scope waits for a trigger. Since we haven't configured one yet, it will wait forever. To manually trigger it, click the *Force* button in the *Trigger* section at the top right. You should see the remainder of the buffer fill, then the screen will display the captured waveforms. The result will look something like the following figure.

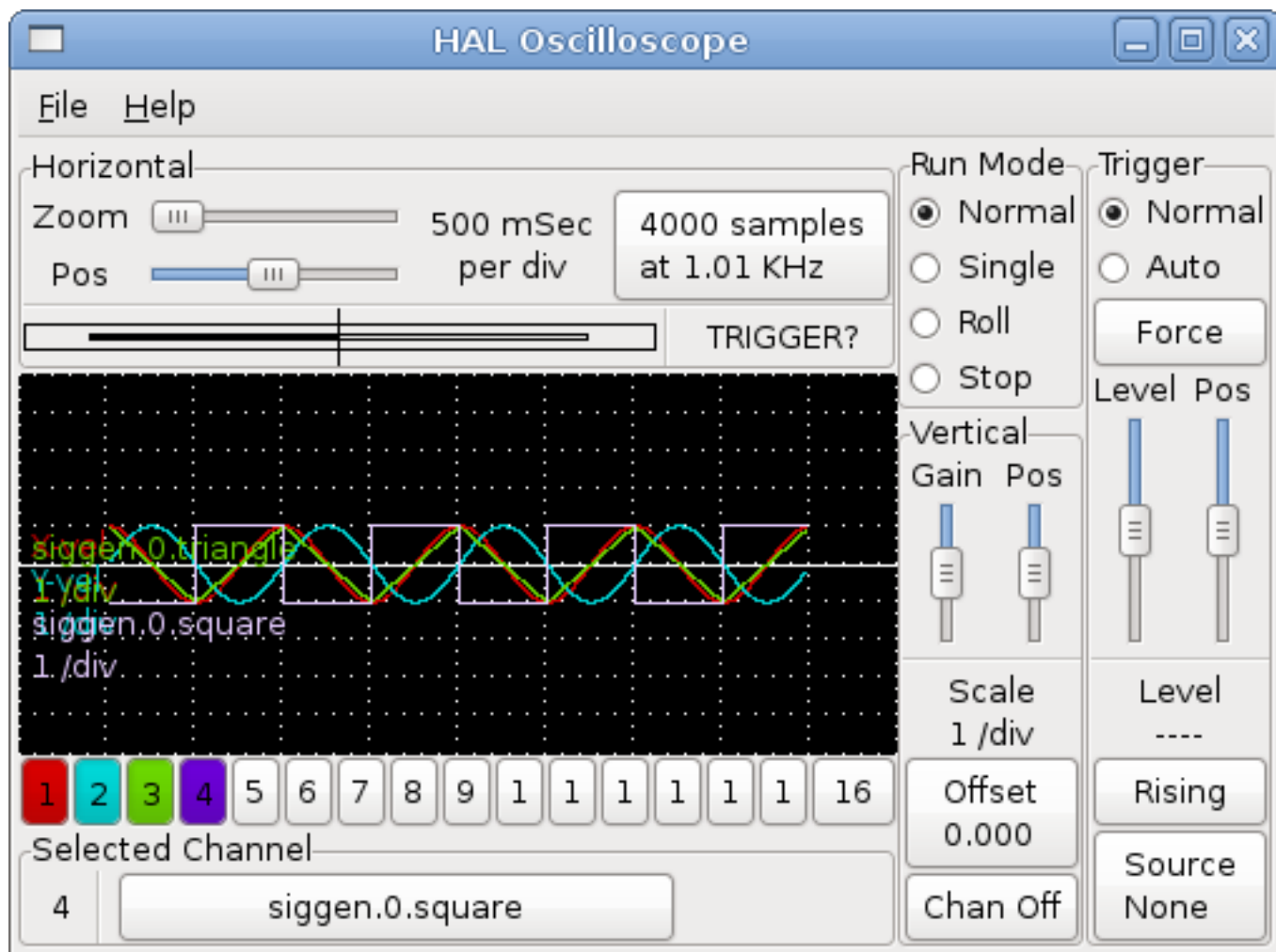


Figure 15.12: Captured Waveforms

The *Selected Channel* box at the bottom tells you that the purple trace is the currently selected one, channel 4, which is displaying the value of the pin *siggen.0.square*. Try clicking channel buttons 1 through 3 to highlight the other three traces.

15.4.6.3 Vertical Adjustments

The traces are rather hard to distinguish since all four are on top of each other. To fix this, we use the *Vertical* controls in the box to the right of the screen. These controls act on the currently selected channel. When adjusting the gain, notice that it covers a huge range - unlike a real scope, this one can display signals ranging from very tiny (pico-units) to very large (Tera-units). The position control moves the displayed trace up and down over the height of the screen only. For larger adjustments the offset button should be used.



Figure 15.13: Vertical Adjustment

The large *Selected Channel* button at the bottom indicates that channel 1 is currently selected channel and that it matches the X-vel signal. Try clicking on the other channels to put their traces in evidence and to be able to move them with the *Pos* cursor.

15.4.6.4 Triggering

Using the *Force* button is a rather unsatisfying way to trigger the scope. To set up real triggering, click on the *Source* button at the bottom right. It will pop up the *Trigger Source* dialog, which is simply a list of all the probes that are currently connected. Select a probe to use for triggering by clicking on it. For this example we will use channel 3, the triangle wave as shown in the following figure.



Figure 15.14: Trigger Source Dialog

After setting the trigger source, you can adjust the trigger level and trigger position using the sliders in the *Trigger* box along the right edge. The level can be adjusted from the top to the bottom of the screen, and is displayed below the sliders. The position is the location of the trigger point within the overall record. With the slider all the way down, the trigger point is at the end of the record, and halscope displays what happened before the trigger point. When the slider is all the way up, the trigger point is at the beginning of the record, displaying what happened after it was triggered. The trigger point is visible as a vertical line in the progress box above the screen. The trigger polarity can be changed by clicking the button just below the trigger level display. It will then become *descendant*. Note that changing the trigger position stops the scope once the position has been adjusted, you relaunch the scope by clicking on the *Normal* button of *Run mode* the group.

Now that we have adjusted the vertical controls and triggering, the scope display looks something like the following figure.

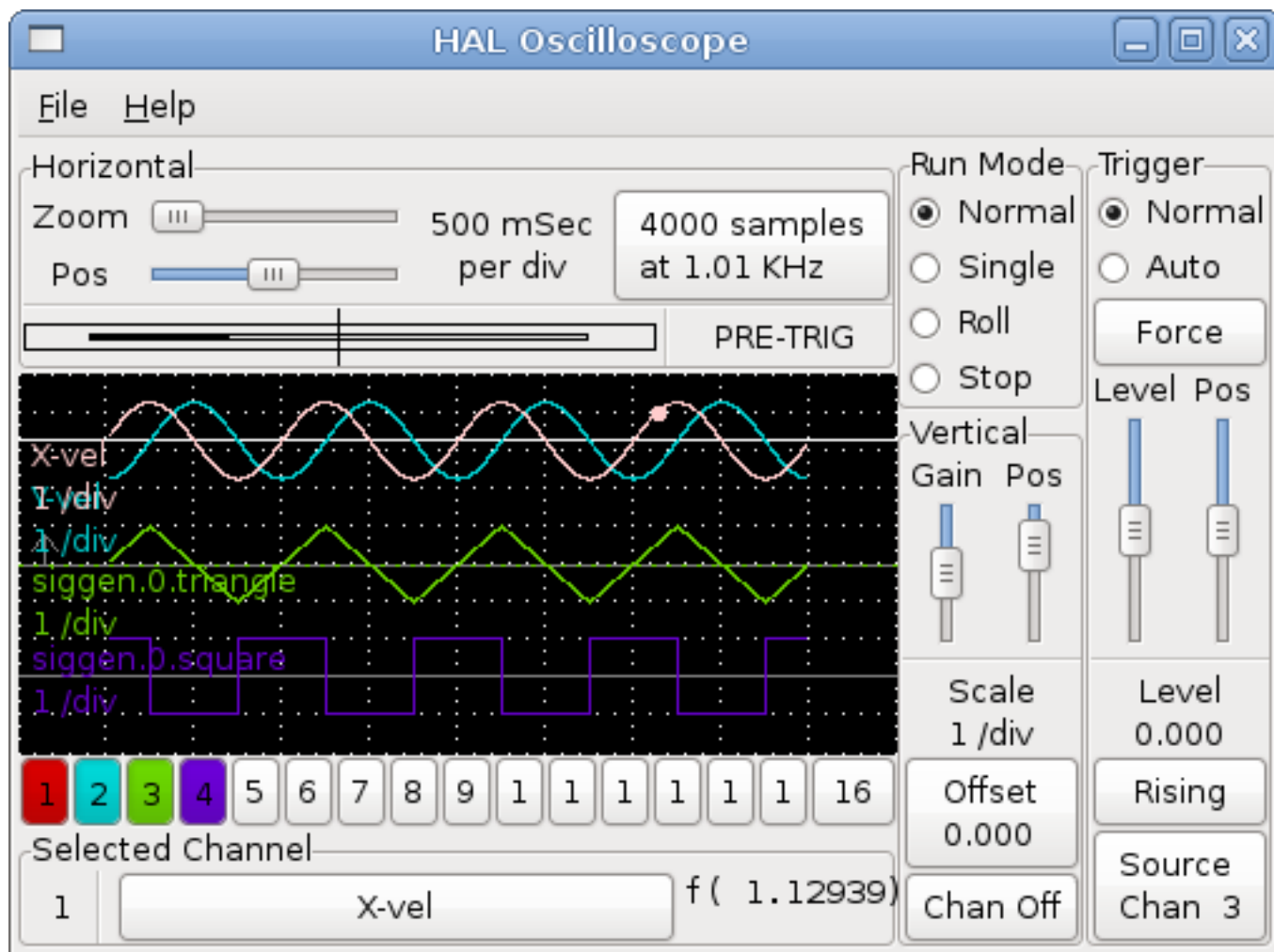


Figure 15.15: Waveforms with Triggering

15.4.6.5 Horizontal Adjustments

To look closely at part of a waveform, you can use the zoom slider at the top of the screen to expand the waveforms horizontally, and the position slider to determine which part of the zoomed waveform is visible. However, sometimes simply expanding the waveforms isn't enough and you need to increase the sampling rate. For example, we would like to look at the actual step pulses that are being generated in our example. Since the step pulses may be only 50 μ s long, sampling at 1 kHz isn't fast enough. To change the sample rate, click on the button that displays the number of samples and sample rate to bring up the *Select Sample Rate* dialog, figure . For this example, we will click on the 50 μ s thread, *fast*, which gives us a sample rate of about 20 kHz. Now instead of displaying about 4 seconds worth of data, one record is 4000 samples at 20 kHz, or about 0.20 seconds.



Figure 15.16: Sample Rate Dialog

15.4.6.6 More Channels

Now let's look at the step pulses. Halscope has 16 channels, but for this example we are using only 4 at a time. Before we select any more channels, we need to turn off a couple. Click on the channel 2 button, then click the *Chan Off* button at the bottom of the *Vertical* box. Then click on channel 3, turn it off, and do the same for channel 4. Even though the channels are turned off, they still remember what they are connected to, and in fact we will continue to use channel 3 as the trigger source. To add new channels, select channel 5, and choose pin *stepgen.0.dir*, then channel 6, and select *stepgen.0.step*. Then click run mode *Normal* to start the scope, and adjust the horizontal zoom to 5 ms per division. You should see the step pulses slow down as the velocity command (channel 1) approaches zero, then the direction pin changes state and the step pulses speed up again. You might want to increase the

gain on channel 1 to about 20 milli per division to better see the change in the velocity command. The result should look like the following figure.

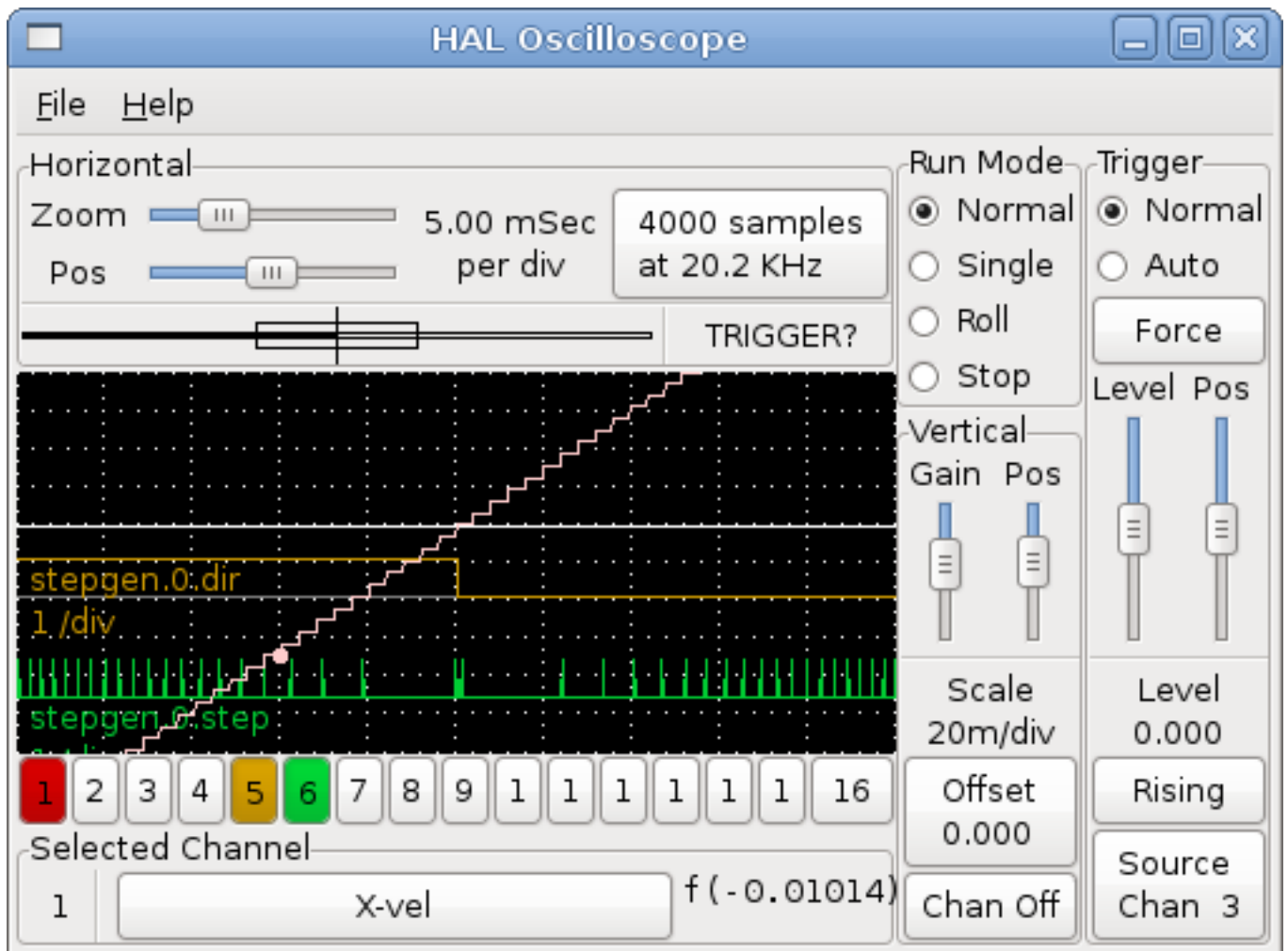


Figure 15.17: Step Pulses

15.4.6.7 More samples

If you want to record more samples at once, restart realtime and load halscope with a numeric argument which indicates the number of samples you want to capture.

```
halcmd loadusr halscope 80000
```

If the *scope_rt* component was not already loaded, halscope will load it and request 80000 total samples, so that when sampling 4 channels at a time there will be 20000 samples per channel. (If *scope_rt* was already loaded, the numeric argument to halscope will have no effect).

15.5 HAL General Reference

15.5.1 HAL Entity Names

All HAL entities are accessible and manipulable by their names, so documenting the names of pins, signals, parameters, etc., is very important. Names in HAL have a maximum length of 41 characters (as defined by `HAL_NAME_LEN` in `hal.h`). Many names will be presented in the general form, with formatted text *<like-this>* representing fields of various values.

When pins, signals, or parameters are described for the first time, their name will be preceded by their type in parentheses (*float*) and followed by a brief description. Typical pins definitions look like these examples:

(bit) parport.<portnum>.pin-<pinnum>-in

The HAL pin associated with the physical input pin *<pinnum>* of the db25 connector.

(float) pid.<loopnum>.output

The PID loop output

Occasionally, an abbreviated version of the name may be used, for example the second pin above could be simply called with *.output* when it can be done without causing confusion.

15.5.2 HAL General Naming Conventions

Consistent naming conventions would make HAL much easier to use. For example, if every encoder driver provided the same set of pins and named them the same way it would be easy to change from one type of encoder driver to another. Unfortunately, like many open-source projects, HAL is a combination of things that were designed, and things that simply evolved. As a result, there are many inconsistencies. This section attempts to address that problem by defining some conventions, but it will probably be a while before all the modules are converted to follow them.

Halcmd and other low-level HAL utilities treat HAL names as single entities, with no internal structure. However, most modules do have some implicit structure. For example, a board provides several functional blocks, each block might have several channels, and each channel has one or more pins. This results in a structure that resembles a directory tree. Even though halcmd doesn't recognize the tree structure, proper choice of naming conventions will let it group related items together (since it sorts the names). In addition, higher level tools can be designed to recognize such structure, if the names provide the necessary information. To do that, all HAL components should follow these rules:

- Dots (".") separate levels of the hierarchy. This is analogous to the slash ("/") in a filename.
- Hyphens ("-") separate words or fields in the same level of the hierarchy.
- HAL components should not use underscores or "MixedCase". ³
- Use only lowercase letters and numbers in names.

15.5.3 Hardware Driver Naming Conventions

Note

Most drivers do not follow these conventions in version 2.0. This chapter is really a guide for future developments.

³Underlined characters have been removed, but there are still a few cases of broken mixture, for example *pid.0.Pgain* in place of *pid.0.p-gain*.

15.5.3.1 Pins/Parameters names

Hardware drivers should use five fields (on three levels) to make up a pin or parameter name, as follows:

```
<device-name>.<device-num>.<io-type>.<chan-num>.<specific-name>
```

The individual fields are:

<device-name>

The device that the driver is intended to work with. This is most often an interface board of some type, but there are other possibilities.

<device-num>

It is possible to install more than one servo board, parallel port, or other hardware device in a computer. The device number identifies a specific device. Device numbers start at 0 and increment.⁴

<io-type>

Most devices provide more than one type of I/O. Even the simple parallel port has both digital inputs and digital outputs. More complex boards can have digital inputs and outputs, encoder counters, pwm or step pulse generators, analog-to-digital converters, digital-to-analog converters, or other unique capabilities. The I/O type is used to identify the kind of I/O that a pin or parameter is associated with. Ideally, drivers that implement the same I/O type, even if for very different devices, should provide a consistent set of pins and parameters and identical behavior. For example, all digital inputs should behave the same when seen from inside the HAL, regardless of the device.

<chan-num>

Virtually every I/O device has multiple channels, and the channel number identifies one of them. Like device numbers, channel numbers start at zero and increment.⁵ If more than one device is installed, the channel numbers on additional devices start over at zero. If it is possible to have a channel number greater than 9, then channel numbers should be two digits, with a leading zero on numbers less than 10 to preserve sort ordering. Some modules have pins and/or parameters that affect more than one channel. For example a PWM generator might have four channels with four independent "duty-cycle" inputs, but one "frequency" parameter that controls all four channels (due to hardware limitations). The frequency parameter should use "0-3" as the channel number.

<specific-name>

An individual I/O channel might have just a single HAL pin associated with it, but most have more than one. For example, a digital input has two pins, one is the state of the physical pin, the other is the same thing inverted. That allows the configurator to choose between active high and active low inputs. For most io-types, there is a standard set of pins and parameters, (referred to as the "canonical interface") that the driver should implement. The canonical interfaces are described in the [Canonical Device Interfaces](#) chapter.

Examples

motenc.0.encoder.2.position

The position output of the third encoder channel on the first Motenc board.

⁴Some hardware use jumpers or other devices to define a specific identification for each. Ideally, the driver provides a way for the user to say, the *device-num 0 is specific to the device that has ID XXX*, its subsets will all carry a number starting with 0. But at present, some drivers use the ID directly as the peripheral number. Which means it is possible to have a device number 2, without having any in number 0. This is a bug that should disappear in version 2.1.

⁵One exception to the "channel numbers start at zero" rule is the parallel port. Its HAL pins are numbered with the corresponding pin number on the DB-25 connector. This is convenient for wiring, but inconsistent with other drivers. There is some debate over whether this is a bug or a feature.

stg.0.din.03.in

The state of the fourth digital input on the first Servo-to-Go board.

ppmc.0.pwm.00-03.frequency

The carrier frequency used for PWM channels 0 through 3 on the first Pico Systems ppmc board.

15.5.3.2 Function Names

Hardware drivers usually only have two kinds of HAL functions, ones that read the hardware and update HAL pins, and ones that write to the hardware using data from HAL pins. They should be named as follows:

```
<device-name>-<device-num>.<io-type>-<chan-num-range>.read|write
```

<device-name>

The same as used for pins and parameters.

<device-num>

The specific device that the function will access.

<io-type>

Optional. A function may access all of the I/O on a board, or it may access only a certain type. For example, there may be independent functions for reading encoder counters and reading digital I/O. If such independent functions exist, the <io-type> field identifies the type of I/O they access. If a single function reads all I/O provided by the board, <io-type> is not used. ⁶

<chan-num-range>

Optional. Used only if the <io-type> I/O is broken into groups and accessed by different functions.

read|write

Indicates whether the function reads the hardware or writes to it.

Examples

motenc.0.encoder.read

Reads all encoders on the first motenc board.

generic8255.0.din.09-15.read

Reads the second 8 bit port on the first generic 8255 based digital I/O board.

ppmc.0.write

Writes all outputs (step generators, pwm, DACs, and digital) on the first Pico Systems ppmc board.

15.6 Core Components

See also the man pages *motion(9)*.

⁶Note to driver programmers: do NOT implement separate functions for different I/O types unless they are interruptible and can work in independent threads. If interrupting an encoder read, reading digital inputs, and then resuming the encoder read will cause problems, then implement a single function that does everything.

15.6.1 Motion

These pins and parameters are created by the realtime *motmod* module.

This module provides a HAL interface for LinuxCNC's motion planner.

Basically *motmod* takes in a list of waypoints and generates a nice blended and constraint-limited stream of joint positions to be fed to the motor drives.

Optionally the number of Digital I/O is set with *num_dio*. The number of Analog I/O is set with *num_aio*, default is 4 each. The number of Spindles is set with *num_spindles*, default is 1.

Pin and parameter names starting with *axis.L* and *joint.N* are read and updated by the motion-controller function.

Motion is loaded with the *motmod* command. A kins should be loaded before motion.

```
loadrt motmod base_period_nsec=['period'] servo_period_nsec=['period']
              traj_period_nsec=['period'] num_joints=['0-9']
              num_dio=['1-64'] num_aio=['1-16'] unlock_joints_mask=['0xNN']
              num_spindles=['1-8']
```

- *base_period_nsec* = 50000 - the *Base* task period in nanoseconds. This is the fastest thread in the machine.

Note

On servo-based systems, there is generally no reason for *base_period_nsec* to be smaller than *servo_period_nsec*. On machines with software step generation, the *base_period_nsec* determines the maximum number of steps per second. In the absence of long step length and step space requirements, the absolute maximum step rate is one step per *base_period_nsec*. Thus, the *base_period_nsec* shown above gives an absolute maximum step rate of 20,000 steps per second. 50,000 ns (50 us) is a fairly conservative value. The smallest usable value is related to the Latency Test result, the necessary step length, and the processor speed. Choosing a *base_period_nsec* that is too low can lead to the "Unexpected real time delay" message, lockups, or spontaneous reboots.

- *servo_period_nsec* = 1000000 - This is the *Servo* task period in nanoseconds. This value will be rounded to an integer multiple of *base_period_nsec*. This period is used even on systems based on stepper motors.

This is the rate at which new motor positions are computed, following error is checked, PID output values are updated, and so on. Most systems will not need to change this value. It is the update rate of the low level motion planner.

- *traj_period_nsec* = 100000 - This is the *Trajectory Planner* task period in nanoseconds. This value will be rounded to an integer multiple of *servo_period_nsec*. Except for machines with unusual kinematics (e.g., hexapods) there is no reason to make this value larger than *servo_period_nsec*.

15.6.1.1 Options

If the number of digital I/O needed is more than the default of 4 you can add up to 64 digital I/O by using the *num_dio* option when loading *motmod*.

If the number of analog I/O needed is more than the default of 4 you can add up to 16 analog I/O by using the *num_aio* option when loading *motmod*.

The *unlock_joints_mask* parameter is used to create pins for a joint used as a locking indexer (typically a rotary). The mask bits select the joint(s). The LSB of the mask selects joint 0. Example:

```
unlock_joints_mask=0x38 selects joints 3,4,5
```

15.6.1.2 Pins

These pins, parameters, and functions are created by the realtime *motmod* module.

- *motion.adaptive-feed* - (float, in) When adaptive feed is enabled with *M52 P1*, the commanded velocity is multiplied by this value. This effect is multiplicative with the NML-level feed override value and *motion.feed-hold*. As of version 2.9 of LinuxCNC it is possible to use a negative adaptive feed value to run the G-code path in reverse.
- *motion.analog-in-00* - (float, in) These pins (00, 01, 02, 03 or more if configured) are controlled by M66.
- *motion.analog-out-00* - (float, out) These pins (00, 01, 02, 03 or more if configured) are controlled by M67 or M68.
- *motion.coord-error* - (bit, out) TRUE when motion has encountered an error, such as exceeding a soft limit
- *motion.coord-mode* - (bit, out) TRUE when motion is in *coordinated mode*, as opposed to *teleop mode*
- *motion.current-vel* - (float, out) The current tool velocity in user units per second.
- *motion.digital-in-00* - (bit, in) These pins (00, 01, 02, 03 or more if configured) are controlled by M62-65.
- *motion.digital-out-00* - (bit, out) These pins (00, 01, 02, 03 or more if configured) are controlled by the M62-65.
- *motion.distance-to-go* - (float,out) The distance remaining in the current move.
- *motion.enable* - (bit, in) If this bit is driven FALSE, motion stops, the machine is placed in the *machine off* state, and a message is displayed for the operator. For normal motion, drive this bit TRUE.
- *motion.feed-hold* - (bit, in) When Feed Stop Control is enabled with *M53 P1*, and this bit is TRUE, the feed rate is set to 0.
- *motion.feed-inhibit* - (bit, in) When this bit is TRUE, the feed rate is set to 0. This will be delayed during spindle synch moves till the end of the move.
- *motion.in-position* - (bit, out) TRUE if the machine is in position.
- *motion.motion-enabled* - (bit, out) TRUE when in *machine on* state.
- *motion.motion-type* - (s32, out) These values are from `src/emc/nml_intf/motion_types.h`
 - 0: Idle (no motion)
 - 1: Traverse
 - 2: Linear feed
 - 3: Arc feed
 - 4: Tool change
 - 5: Probing
 - 6: Rotary axis indexing
- *motion.on-soft-limit* - (bit, out) TRUE when the machine is on a soft limit.
- *motion.probe-input* - (bit, in) *G38.n* uses the value on this pin to determine when the probe has made contact. TRUE for probe contact closed (touching), FALSE for probe contact open.

- *motion.program-line* - (s32, out) The current program line while executing. Zero if not running or between lines while single stepping.
- *motion.requested-vel* - (float, out) The current requested velocity in user units per second. This value is the F-word setting from the G-code file, possibly reduced to accommodate machine velocity and acceleration limits. The value on this pin does not reflect the feed override or any other adjustments.
- *motion.teleop-mode* - (bit, out) TRUE when motion is in *teleop mode*, as opposed to *coordinated mode*
- *motion.tooloffset.x ... motion.tooloffset.w* - (float, out, one per axis) shows the tool offset in effect; it could come from the tool table (*G43* active), or it could come from the gcode (*G43.1* active)
- *motion.on-soft-limit* - (bit, out) TRUE when the machine is on a soft limit.
- *motion.probe-input* - (bit, in) *G38.n* uses the value on this pin to determine when the probe has made contact. TRUE for probe contact closed (touching), FALSE for probe contact open.
- *motion.program-line* - (s32, out) The current program line while executing. Zero if not running or between lines while single stepping.
- *motion.requested-vel* - (float, out) The current requested velocity in user units per second. This value is the F-word setting from the G-code file, possibly reduced to accommodate machine velocity and acceleration limits. The value on this pin does not reflect the feed override or any other adjustments.
- *motion.teleop-mode* - (bit, out) TRUE when motion is in *teleop mode*, as opposed to *coordinated mode*
- *motion.tooloffset.x ... motion.tooloffset.w* - (float, out, one per axis) shows the tool offset in effect; it could come from the tool table (*G43* active), or it could come from the G-code (*G43.1* active)

15.6.1.3 Parameters

Many of these parameters serve as debugging aids, and are subject to change or removal at any time.

- *motion-command-handler.time* - (s32, RO)
- *motion-command-handler.tmax* - (s32, RW)
- *motion-controller.time* - (s32, RO)
- *motion-controller.tmax* - (s32, RW)
- *motion.debug-bit-0* - (bit, RO) This is used for debugging purposes.
- *motion.debug-bit-1* - (bit, RO) This is used for debugging purposes.
- *motion.debug-float-0* - (float, RO) This is used for debugging purposes.
- *motion.debug-float-1* - (float, RO) This is used for debugging purposes.
- *motion.debug-float-2* - (float, RO) This is used for debugging purposes.
- *motion.debug-float-3* - (float, RO) This is used for debugging purposes.
- *motion.debug-s32-0* - (s32, RO) This is used for debugging purposes.
- *motion.debug-s32-1* - (s32, RO) This is used for debugging purposes.
- *motion.servo.last-period* - (u32, RO) The number of CPU cycles between invocations of the servo thread. Typically, this number divided by the CPU speed gives the time in seconds, and can be used to determine whether the realtime motion controller is meeting its timing constraints
- *motion.servo.last-period-ns* - (float, RO)

15.6.1.4 Functions

Generally, these functions are both added to the servo-thread in the order shown.

- *motion-command-handler* - Processes motion commands coming from user space
- *motion-controller* - Runs the LinuxCNC motion controller

15.6.2 Spindle

LinuxCNC can control upto eight spindles. Motion will produce the following pins: The *N* (integer between 0 and 7) substitutes the spindle number.

15.6.2.1 Pins

- *spindle.N.at-speed* - (bit, in) Motion will pause until this pin is TRUE, under the following conditions:
 - before the first feed move after each spindle start or speed change;
 - before the start of every chain of spindle-synchronized moves;
 - and if in CSS mode, at every rapid to feed transition. This input can be used to ensure that the spindle is up to speed before starting a cut, or that a lathe spindle in CSS mode has slowed down after a large to small facing pass before starting the next pass at the large diameter. Many VFDs have an *at speed* output. Otherwise, it is easy to generate this signal with the *HAL near* component, by comparing requested and actual spindle speeds.
 - *spindle.N.brake* - (bit, out) TRUE when the spindle brake should be applied.
 - *spindle.N.forward* - (bit, out) TRUE when the spindle should rotate forward.
 - *spindle.N.index-enable* - (bit, I/O) For correct operation of spindle synchronized moves, this pin must be hooked to the index-enable pin of the spindle encoder.
 - *spindle.N.inhibit* - (bit, in) When this bit is TRUE, the spindle speed is set to 0.
 - *spindle.N.on* - (bit, out) TRUE when spindle should rotate.
 - *spindle.N.reverse* - (bit, out) TRUE when the spindle should rotate backward
 - *spindle.N.revs* - (float, in) For correct operation of spindle synchronized moves, this signal must be hooked to the position pin of the spindle encoder. The spindle encoder position should be scaled such that spindle-revs increases by 1.0 for each rotation of the spindle in the clockwise (*M3*) direction.
 - *spindle.N.speed-in* - (float, in) Feedback of actual spindle speed in rotations per second. This is used by feed-per-revolution motion (*G95*). If your spindle encoder driver does not have a velocity output, you can generate a suitable one by sending the spindle position through a *ddt* component. If you do not have a spindle encoder, you can loop back *spindle.N.speed-out-rps*.
 - *spindle.N.speed-out* - (float, out) Commanded spindle speed in rotations per minute. Positive for spindle forward (*M3*), negative for spindle reverse (*M4*).
 - *spindle.N.speed-out-abs* - (float, out) Commanded spindle speed in rotations per minute. This will always be a positive number.
 - *spindle.N.speed-out-rps* - (float, out) Commanded spindle speed in rotations per second. Positive for spindle forward (*M3*), negative for spindle reverse (*M4*).
 - *spindle.N.speed-out-rps-abs* - (float, out) Commanded spindle speed in rotations per second. This will always be a positive number.
-

- *spindle.N.orient-angle* - (float,out) Desired spindle orientation for M19. Value of the M19 R word parameter plus the value of the [RS274NGC]ORIENT_OFFSET INI parameter.
- *spindle.N.orient-mode* - (s32,out) Desired spindle rotation mode M19. Default 0.
- *spindle.N.orient* - (out,bit) Indicates start of spindle orient cycle. Set by M19. Cleared by any of M3, M4, or M5. If spindle-orient-fault is not zero during spindle-orient true, the M19 command fails with an error message.
- *spindle.N.is-oriented* - (in, bit) Acknowledge pin for spindle-orient. Completes orient cycle. If spindle-orient was true when spindle-is-oriented was asserted, the spindle-orient pin is cleared and the spindle-locked pin is asserted. Also, the spindle-brake pin is asserted.
- *spindle.N.orient-fault* - (s32, in) Fault code input for orient cycle. Any value other than zero will cause the orient cycle to abort.
- *spindle.N.lock* - (bit, out) Spindle orient complete pin. Cleared by any of M3, M4, or M5.

HAL pin usage for M19 orient spindle Conceptually the spindle is in one of the following modes:

- rotation mode (the default)
- searching for desired orientation mode
- orientation complete mode.

When an M19 is executed, the spindle changes to *searching for desired orientation*, and the spindle.N.orient HAL pin is asserted. The desired target position is specified by the spindle.N.orient-angle and spindle.N.orient-fwd pins and driven by the M19 R and P parameters.

The HAL support logic is expected to react to spindle.N.orient by moving the spindle to the desired position. When this is complete, the HAL logic is expected to acknowledge this by asserting the spindle.N.is-oriented pin.

Motion then acknowledges this by deasserting the spindle.N.orient pin and asserts the spindle.N.locked pin to indicate *orientation complete* mode. It also raises the spindle.N.brake pin. The spindle now is in *orientation complete* mode.

If, during spindle.N.orient being true, and spindle.N.is-oriented not yet asserted the spindle.N.orient pin has a value other than zero, the M19 command is aborted, a message including the fault code is displayed, and the motion queue is flushed. The spindle reverts to rotation mode.

Also, any of the M3, M4 or M5 commands cancel either *searching for desired orientation* or *orientation complete* mode. This is indicated by deasserting both the spindle-orient and spindle-locked pins.

The spindle-orient-mode pin reflects the M19 P word and shall be interpreted as follows:

- 0: rotate clockwise or counterclockwise for smallest angular movement
- 1: always rotate clockwise
- 2: always rotate counterclockwise

It can be used with the orient HAL component which provides a PID command value based on spindle encoder position, spindle-orient-angle and spindle-orient-mode.

15.6.3 Axis and Joint Pins and Parameters

These pins and parameters are created by the realtime *motmod* module. [In *trivial kinematics* machines, there is a one-to-one correspondence between joints and axes.] They are read and updated by the *motion-controller* function.

See the motion man page *motion(9)* for details on the pins and parameters.

15.6.4 iocontrol

iocontrol - accepts NML I/O commands, interacts with HAL in userspace.

The signals are turned on and off in userspace - if you have strict timing requirements or simply need more i/o, consider using the realtime synchronized i/o provided by [motion](#) instead.

15.6.4.1 Pins

- *iocontrol.0.coolant-flood* - (bit, out) TRUE when flood coolant is requested.
- *iocontrol.0.coolant-mist* - (bit, out) TRUE when mist coolant is requested.
- *iocontrol.0.emc-enable-in* - (bit, in) Should be driven FALSE when an external E-Stop condition exists.
- *iocontrol.0.lube* - (bit, out) TRUE when lube is commanded.
- *iocontrol.0.lube_level* - (bit, in) Should be driven TRUE when lube level is high enough.
- *iocontrol.0.tool-change* - (bit, out) TRUE when a tool change is requested.
- *iocontrol.0.tool-changed* - (bit, in) Should be driven TRUE when a tool change is completed.
- *iocontrol.0.tool-number* - (s32, out) The current tool number.
- *iocontrol.0.tool-prepare-number* - (s32, out) The number of the next tool, from the RS274NGC T-word.
- *iocontrol.0.tool-prepare* - (bit, out) TRUE when a tool prepare is requested.
- *iocontrol.0.tool-prepared* - (bit, in) Should be driven TRUE when a tool prepare is completed.
- *iocontrol.0.user-enable-out* - (bit, out) FALSE when an internal E-Stop condition exists.
- *iocontrol.0.user-request-enable* - (bit, out) TRUE when the user has requested that E-Stop be cleared.

15.6.5 INI settings

A number of INI settings are made available as HAL input pins.

15.6.5.1 Pins

N refers to a joint number, L refers to an axis letter

- *ini.N.ferror* - (float, in) [JOINT_N]FERROR
 - *ini.N.min_ferror* - (float, in) [JOINT_N]MIN_FERROR
 - *ini.N.backlash* - (float, in) [JOINT_N]BACKLASH
 - *ini.N.min_limit* - (float, in) [JOINT_N]MIN_LIMIT
 - *ini.N.max_limit* - (float, in) [JOINT_N]MAX_LIMIT
 - *ini.N.max_velocity* - (float, in) [JOINT_N]MAX_VELOCITY
 - *ini.N.max_acceleration* - (float, in) [JOINT_N]MAX_ACCELERATION
 - *ini.N.home* - (float, in) [JOINT_N]HOME
 - *ini.N.home_offset* - (float, in) [JOINT_N]HOME_OFFSET
-

- *ini.N.home_offset* - (s32, in) [JOINT_N]HOME_SEQUENCE
- *ini.L.min_limit* - (float, in) [AXIS_L]MIN_LIMIT
- *ini.L.max_limit* - (float, in) [AXIS_L]MAX_LIMIT
- *ini.L.max_velocity* - (float, in) [AXIS_L]MAX_VELOCITY
- *ini.L.max_acceleration* - (float, in) [AXIS_L]MAX_ACCELERATION

Note

The per-axis min_limit and max_limit pins are honored continuously after homing. The per-axis ferror and min_ferror pins are honored when the machine is on and not in position. The per-axis max_velocity and max_acceleration pins are sampled when the machine is on and the motion_state is free (homing or jogging) but are not sampled when in a program is running (auto mode) or in MDI mode. Consequently, changing the pin values when a program is running will not have effect until the program is stopped and the motion_state is again free.

- *ini.traj_arc_blend_enable* - (bit, in) [TRAJ]ARC_BLEND_ENABLE
- *ini.traj_arc_blend_fallback_enable* - (bit, in) [TRAJ]ARC_BLEND_FALLBACK_ENABLE
- *ini.traj_arc_blend_gap_cycles* - (float, in) [TRAJ]ARC_BLEND_GAP_CYCLES
- *ini.traj_arc_blend_optimization_depth* - (float, in) [TRAJ]ARC_BLEND_OPTIMIZATION_DEPTH
- *ini.traj_arc_blend_ramp_freq* - (float, in) [TRAJ]ARC_BLEND_RAMP_FREQ

Note

The traj_arc_blend pins are sampled continuously but changing pin values while a program is running may not have immediate effect due to queueing of commands.

- *ini.traj_default_acceleration* - (float, in) [TRAJ]DEFAULT_ACCELERATION
- *ini.traj_default_velocity* - (float, in) [TRAJ]DEFAULT_VELOCITY
- *ini.traj_max_acceleration* - (float, in) [TRAJ]MAX_ACCELERATION

15.7 Canonical Device Interfaces

15.7.1 Introduction

The following sections show the pins, parameters, and functions that are supplied by "canonical devices". All HAL device drivers should supply the same pins and parameters, and implement the same behavior.

Note that only the <io-type> and <specific-name> fields are defined for a canonical device. The <device-name>, <device-num>, and <chan-num> fields are set based on the characteristics of the real device.

15.7.2 Digital Input

The canonical digital input (I/O type field: digin) is quite simple.

15.7.2.1 Pins

(bit) in

State of the hardware input.

(bit) in-not

Inverted state of the input.

15.7.2.2 Parameters

None

15.7.2.3 Functions

(funct) read

Read hardware and set **in** and **in-not** HAL pins.

15.7.3 Digital Output

The canonical digital output (I/O type field: **digout**) is also very simple.

15.7.3.1 Pins

(bit) out

Value to be written (possibly inverted) to the hardware output.

15.7.3.2 Parameters

(bit) invert

If TRUE, **out** is inverted before writing to the hardware.

15.7.3.3 Functions

(funct) write

Read **out** and **invert**, and set hardware output accordingly.

15.7.4 Analog Input

The canonical analog input (I/O type: **adcin**). This is expected to be used for analog to digital converters, which convert e.g. voltage to a continuous range of values.

15.7.4.1 Pins

(float) value

The hardware reading, scaled according to the **scale** and **offset** parameters.

value = ((input reading, in hardware-dependent units) * **scale**) - **offset**

15.7.4.2 Parameters

(float) scale

The input voltage (or current) will be multiplied by **scale** before being output to **value**.

(float) offset

This will be subtracted from the hardware input voltage (or current) after the scale multiplier has been applied.

(float) bit_weight

The value of one least significant bit (LSB). This is effectively the granularity of the input reading.

(float) hw_offset

The value present on the input when 0 volts is applied to the input pin(s).

15.7.4.3 Functions

(funct) read

Read the values of this analog input channel. This may be used for individual channel reads, or it may cause all channels to be read

15.7.5 Analog Output

The canonical analog output (I/O Type: **adcout**). This is intended for any kind of hardware that can output a more-or-less continuous range of values. Examples are digital to analog converters or PWM generators.

15.7.5.1 Pins

(float) value

The value to be written. The actual value output to the hardware will depend on the scale and offset parameters.

(bit) enable

If false, then output 0 to the hardware, regardless of the **value** pin.

15.7.5.2 Parameters

(float) offset

This will be added to the **value** before the hardware is updated.

(float) scale

This should be set so that an input of 1 on the **value** pin will cause the analog output pin to read 1 volt.

(float) high_limit (optional)

When calculating the value to output to the hardware, if **value** + **offset** is greater than **high_limit**, then **high_limit** will be used instead.

(float) low_limit (optional)

When calculating the value to output to the hardware, if **value** + **offset** is less than **low_limit**, then **low_limit** will be used instead.

(float) bit_weight (optional)

The value of one least significant bit (LSB), in volts (or mA, for current outputs).

(float) hw_offset (optional)

The actual voltage (or current) that will be output if 0 is written to the hardware.

15.7.5.3 Functions

(funct) write

This causes the calculated value to be output to the hardware. If enable is false, then the output will be 0, regardless of **value**, **scale**, and **offset**. The meaning of "0" is dependent on the hardware. For example, a bipolar 12-bit A/D may need to write 0x1FF (mid scale) to the D/A get 0 volts from the hardware pin. If enable is true, read scale, offset and value and output to the adc (**scale** * **value**) + **offset**. If enable is false, then output 0.

15.8 HAL Tools

15.8.1 Halcmd

Halcmd is a command line tool for manipulating the HAL. There is a rather complete man page for [halcmd](#), which will be installed if you have installed LinuxCNC from either source or a package. The manpage provides usage info:

```
man halcmd
```

If you have compiled LinuxCNC for "run-in-place", you must source the rip-environment script to make the man page available:

```
cd toplevel_directory_for_rip_build
. scripts/rip-environment
man halcmd
```

The [HAL Tutorial](#) has a number of examples of halcmd usage, and is a good tutorial for halcmd.

15.8.2 Halmeter

Halmeter is a *voltmeter* for the HAL. It lets you look at a pin, signal, or parameter, and displays the current value of that item. It is pretty simple to use. Start it by typing **halmeter** in an X windows shell. Halmeter is a GUI application. It will pop up a small window, with two buttons labeled *Select* and *Exit*. Exit is easy - it shuts down the program. Select pops up a larger window, with three tabs. One tab lists all the pins currently defined in the HAL. The next lists all the signals, and the last tab lists all the parameters. Click on a tab, then click on a pin/signal/parameter. Then click on *OK*. The lists will disappear, and the small window will display the name and value of the selected item. The display is updated approximately 10 times per second. If you click *Accept* instead of *OK*, the small window will display the name and value of the selected item, but the large window will remain on the screen. This is convenient if you want to look at a number of different items quickly.

You can have many halmeters running at the same time, if you want to monitor several items. If you want to launch a halmeter without tying up a shell window, type *halmeter &* to run it in the background. You can also make halmeter start displaying a specific item immediately, by adding *pin|sig|par[am]* *<name>* to the command line. It will display the pin, signal, or parameter *<name>* as soon as it starts. (If there is no such item, it will simply start normally.) And finally, if you specify an item to display, you can add *-s* before the pin|sig|param to tell halmeter to use a small window. The item name will be displayed in the title bar instead of under the value, and there will be no buttons. Useful when you want a lot of meters in a small amount of screen space.

Refer to [Halmeter Tutorial](#) section for more information.

Halmeter can be loaded from a terminal or from Axis. Halmeter is faster than Halshow at displaying values. Halmeter has two windows, one to pick the pin, signal, or parameter to monitor and one that displays the value. Multiple Halmeters can be open at the same time. If you use a script to open multiple Halmeters you can set the position of each one with *-g X Y* relative to the upper left corner of your screen. For example:

```
loadusr halmeter pin hm2.0.stepgen.00.velocity-fb -g 0 500
```

See the man page for more options. See section [Halmeter](#).

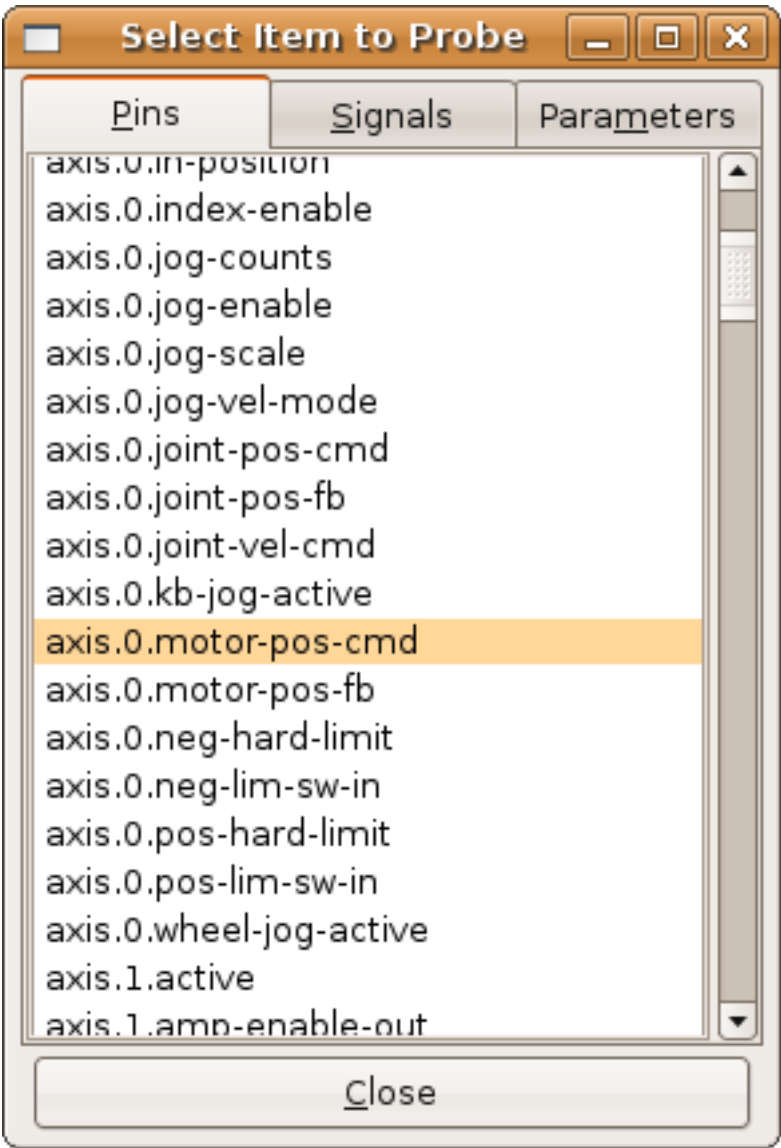


Figure 15.18: Halmeter selection window



Figure 15.19: Halmeter watch window

15.8.3 Halshow

Halshow ([complete usage description](#)) can be started from the command line to show details for selected components, pins, parameters, signals, functions, and threads of a running HAL. The WATCH tab provides a continuous display of selected pin, parameters, and signal items. The File menu provides buttons to 1) save the watch items to a watch list and to load an existing watch list. The watch list items can also be loaded automatically on startup. For command line usage:

```
halshow --help
```

Usage:

```
halshow [Options] [watchfile]
```

Options:

```
--help      (this help)
--fformat format_string_for_float
--iformat format_string_for_int
```

Notes:

Create watchfile in halshow using: 'File/Save Watch List'
linuxcnc must be running for standalone usage

LinuxCNC must be running for standalone usage.

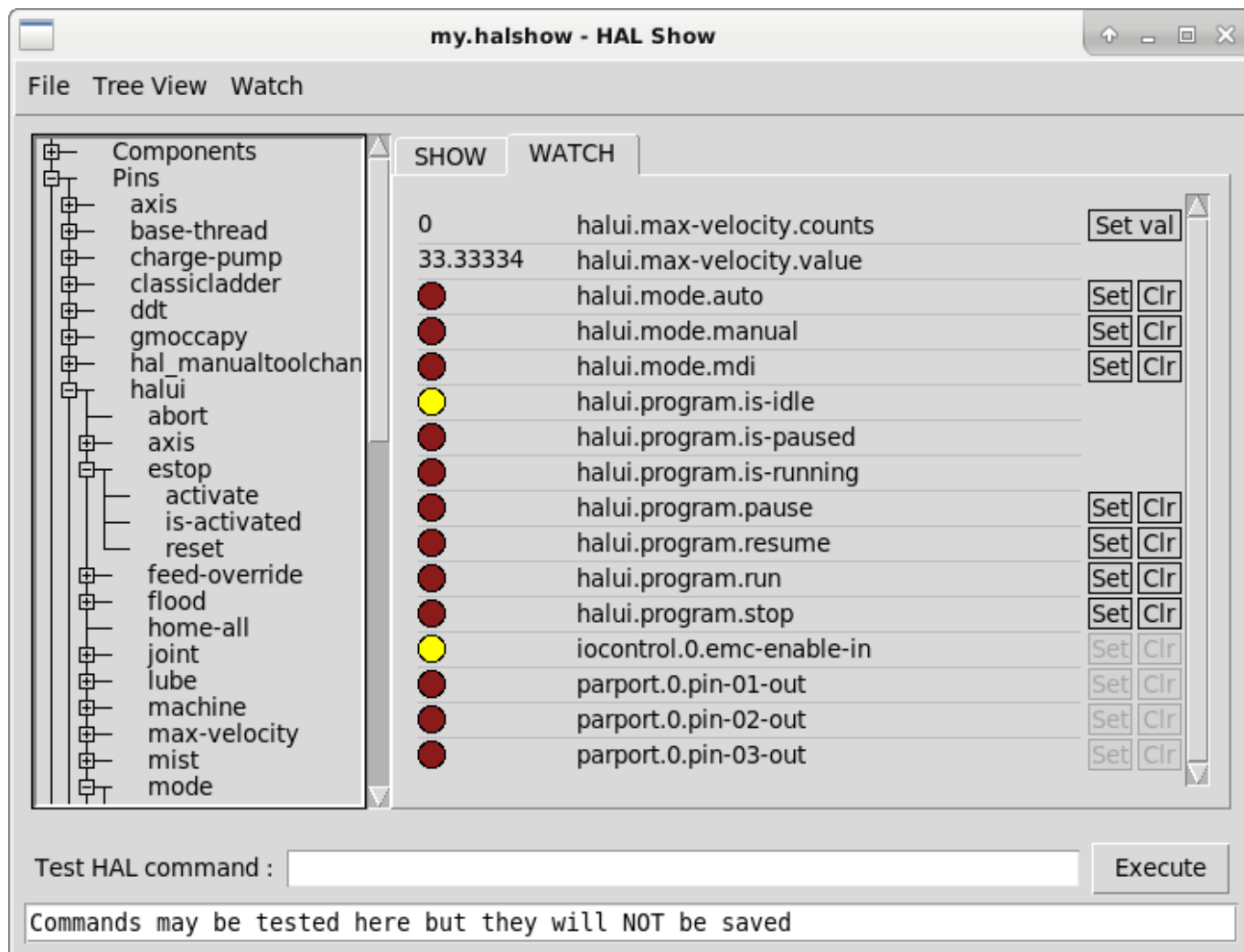


Figure 15.20: Halshow Watch Tab

A watchfile created using the *File/Save Watch List* menu item is formatted as a single line with tokens "pin+", "param+", "sig=+" followed by the appropriate pin, param, or signal name. The token-name pairs are separated by a space character.

Single Line Watchfile Example

```
pin+joint.0.pos-hard-limit pin+joint.1.pos-hard-limit sig+estop-loop
```

A watchfile created using the *File/Save Watch List (multiline)* menu item is formatted with separate lines for each item identified with token-name pairs as described above.

Separated Lines Watchfile Example

```
pin+joint.0.pos-hard-limit
pin+joint.1.pos-hard-limit
sig+estop-loop
```

When loading a watchfile with the *File/Load Watch List* menu item, the token-name pairs may appear as single or multiple lines. Blank lines and lines beginning with a # character are ignored.

15.8.4 Halscope

Halscope is an *oscilloscope* for the HAL. It lets you capture the value of pins, signals, and parameters as a function of time. Complete operating instructions should be located here eventually. For now, refer to section Section 15.4.6 in the tutorial chapter, which explains the basics.

The halscope File menu selector provides buttons to save a configuration or open a previously saved configuration. When halscope is terminated, the last configuration is saved in a file named autosave.halscope.

Configuration files may also be specified when starting halscope from the commandline. Commandline help (-h) usage:

```
halscope -h
Usage:
  halscope [-h] [-i infile] [-o outfile] [num_samples]
```

15.8.5 Sim Pin

sim_pin is a command line utility to display and update any number of writable pins, parameters or signals.

sim_pin Usage

```
Usage:
  sim_pin [Options] name1 [name2 ...] &

Options:
  --help                (this text)
  --title title_string  (window title, default: sim_pin)

Note: LinuxCNC (or a standalone Hal application) must be running
A named item can specify a pin, param, or signal
The item must be writable, e.g.:
  pin:    IN or I/O (and not connected to a signal with a writer)
  param:  RW
  signal: connected to a writable pin

Hal item types bit,s32,u32,float are supported
```

When a bit item is specified, a pushbutton is created to manage the item in one of three manners specified by radio buttons:

- toggle: Toggle value when button pressed
- pulse: Pulse item to 1 once when button pressed
- hold: Set to 1 while button pressed

The bit pushbutton mode can be specified on the command line by formatting the item name:

```
namei/mode=[toggle | pulse | hold]
```

If the mode begins with an uppercase letter, the radio buttons for selecting other modes are not shown

For complete information, see the man page:

```
man sim_pin
```

sim_pin Example (with LinuxCNC running)

```
halcmd loadrt mux2 names=example; halcmd net sig_example example.in0
sim_pin example.sel example.in1 sig_example &
```

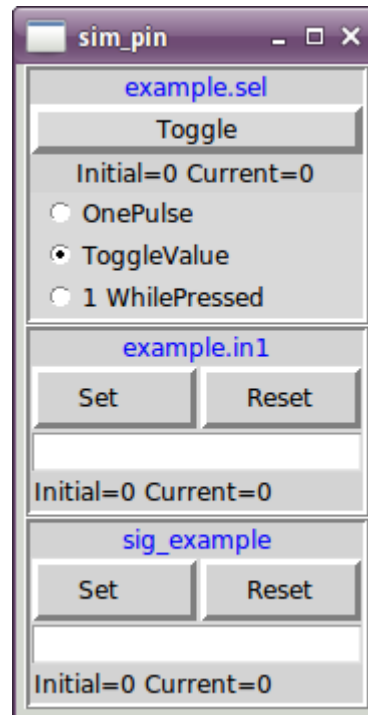


Figure 15.21: sim_pin Window

15.8.6 Simulate Probe

simulate_probe is a simple gui to simulate activation of the pin motion.probe-input. Usage:

```
simulate_probe &
```

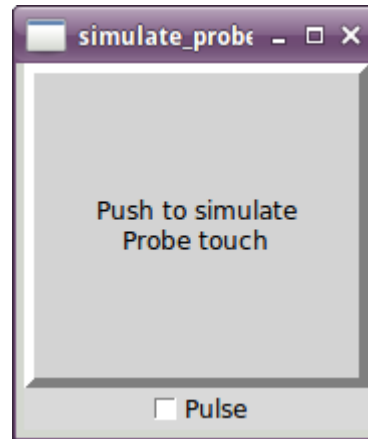


Figure 15.22: simulate_probe Window

15.8.7 Hal Histogram

hal-histogram is a command line utility to display histograms for hal pins.

hal-histogram Usage

Usage:

```
hal-histogram --help | -?
```

or

```
hal-histogram [Options] [pinname]
```

Options:

```
--minvalue  minvalue (minimum bin, default: 0)
--binsize    binsize  (binsize, default: 100)
--nbins      nbins    (number of bins, default: 50)

--logscale   0|1      (y axis log scale, default: 1)
--text       note     (text display, default: "" )
--show       (show count of undisplayed nbins, default off)
--verbose    (progress and debug, default off)
```

Notes:

- 1) LinuxCNC (or another Hal application) must be running
- 2) If no pinname is specified, default is: motion-command-handler.time
- 3) This app may be opened for 5 pins
- 4) pintypes float, s32, u32, bit are supported
- 5) The pin must be associated with a thread supporting floating point
For a base thread, this may require using:
loadrt motmod ... base_thread_fp=1

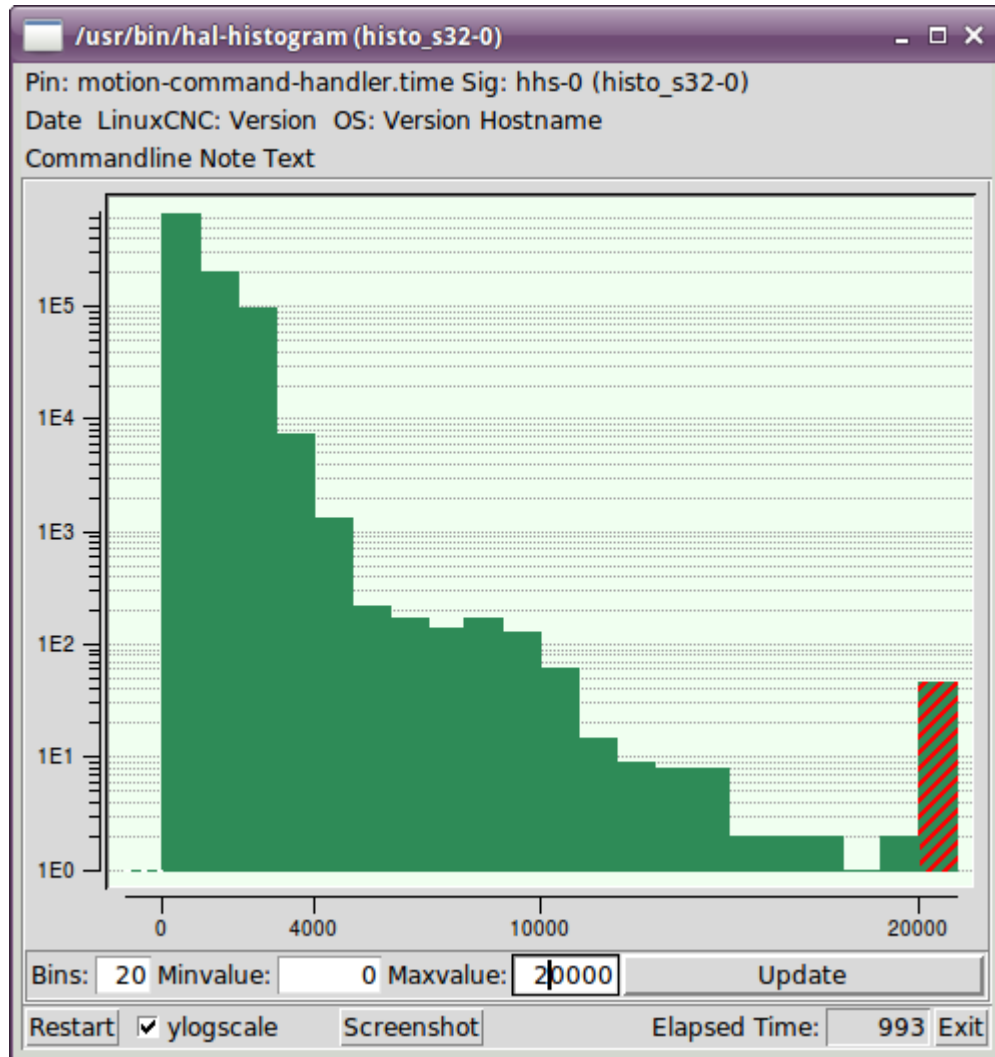


Figure 15.23: hal-histogram Window

15.8.8 Halreport

halreport is a command-line utility that generates a report about hal connections for a running LinuxCNC (or other hal) application. The report shows all signal connections and flags potential problems. Information included:

1. System description and kernel version.
2. Signals and all connected output, io, and input pins.
3. Each pin's component_function, thread, and addf-order.
4. Userspace component pins having non-ordered functions.
5. Identification of unknown functions for unhandled components.
6. Signals with no output.
7. Signals with no inputs.
8. Functions with no addf.

9. Warning tags for components marked as deprecated/obsolete in docs.

10. Real names for pins that use alias names.

The report can be generated from the command line and directed to an output file (or stdout if no outfilename is specified):

halreport Usage

```
Usage:
  halreport -h | --help (this help)
or
  halreport [outfilename]
```

To generate the report for every LinuxCNC startup, include halreport and an output filename as an [APPLICATIONS]APP entry in the ini file.

halreport Example

```
[APPLICATIONS]
APP = halreport /tmp/halreport.txt
```

The function addf-ordering can be important for servo loops where the sequence of the functions computed at each servo period is important. Typically, the order is: read input pins, do the motion command-handler and motion-controller functions, perform pid calculations, and finally write output pins.

For each signal in a critical path, the addf-order of the output pin should be numerically lower than the addf-order of the critical input pins that it connects to.

For routine signal paths that handle switch inputs, user-space pins, etc., the addf-ordering is often not critical. Moreover, the timing of user-space pin value changes cannot be controlled or guaranteed at the intervals typically employed for hal threads.

Example report file excerpts showing a pid loop for a hostmot2 stepgen operated in velocity mode on a trivkins machine with joint.0 corresponding to the X axis coordinate:

```
SIG:    pos-fb-0
  OUT:   h.00.position-fb          hm2_7i92.0.read      servo-thread 001
        (=hm2_7i92.0.stepgen.00.position-fb)
  IN:    X_pid.feedback            X_pid.do-pid-calcs   servo-thread 004
  IN:    joint.0.motor-pos-fb      motion-command-handler servo-thread 002
        .....                  motion-controller      servo-thread 003
...
SIG:    pos-cmd-0
  OUT:   joint.0.motor-pos-cmd      motion-command-handler servo-thread 002
        .....                  motion-controller      servo-thread 003
  IN:    X_pid.command             X_pid.do-pid-calcs   servo-thread 004
...
SIG:    motor-cmd-0
  OUT:   X_pid.output              X_pid.do-pid-calcs   servo-thread 004
  IN:    h.00.velocity-cmd         hm2_7i92.0.write     servo-thread 008
        (=hm2_7i92.0.stepgen.00.velocity-cmd)
```

In the example above, the HALFILE uses halcmd aliases to simplify pin names for an hostmot2 fpga board with commands like:

```
alias pin hm2_7i92.0.stepgen.00.position-fb h.00.position-fb
```

Note

Questionable component function detection may occur for 1) unsupported (deprecated) components, 2) user-created components that use multiple functions or unconventional function naming, or 3) gui-created userspace components that lack distinguishing characteristics such as a prefix based on the gui program name. Questionable functions are tagged with a question mark "?".

Note

Component pins that cannot be associated with a known thread function report the function as "Unknown".

Note

halreport generates a connections report for a running hal application to aid in designing and verifying connections. Pin types and current values are not shown. For this information use applications like halshow, halmeter, halscope or the *show* command available with command-line halcmd program.

15.9 Halshow

The script halshow can help you find your way around a running HAL. This is a very specialized system and it must connect to a working HAL. It cannot run standalone because it relies on the ability of HAL to report what it knows of itself through the halcmd interface library. It is discovery based. Each time halshow runs with a different LinuxCNC configuration it will be different.

As we will soon see, this ability of HAL to document itself is one key to making an effective CNC system.

15.9.1 Starting Halshow

Halshow is in the AXIS menu under Machine/Show HAL Configuration.

Halshow is in the TkLinuxCNC menu under Scripts/HAL Show.

Halshow can be started from a terminal command line and specify formats for integer and float items (pins or signals) and identify a saved watchlist file to use:

```
$ halshow --help
Usage:
  halshow [Options] [watchfile]
Options:
  --help      (this help)
  --fformat format_string_for_float
  --iformat format_string_for_int

Notes:
  Create watchfile in halshow using: 'File/Save Watch List'
  linuxcnc must be running for standalone usage
```

Example to limit number of decimal points for floats and use a file named my.halshow in the current directory:

```
$ halshow --fformat "%.5f" ./my.halshow
```

For more information regarding the format, please refer to <https://www.tcl.tk/man/tcl/TclCmd/format.html>

15.9.2 HAL Tree Area

At the left of its display as shown in figure is a tree view, somewhat like you might see with some file browsers. At the right is a tabbed notebook with tabs for show and watch.

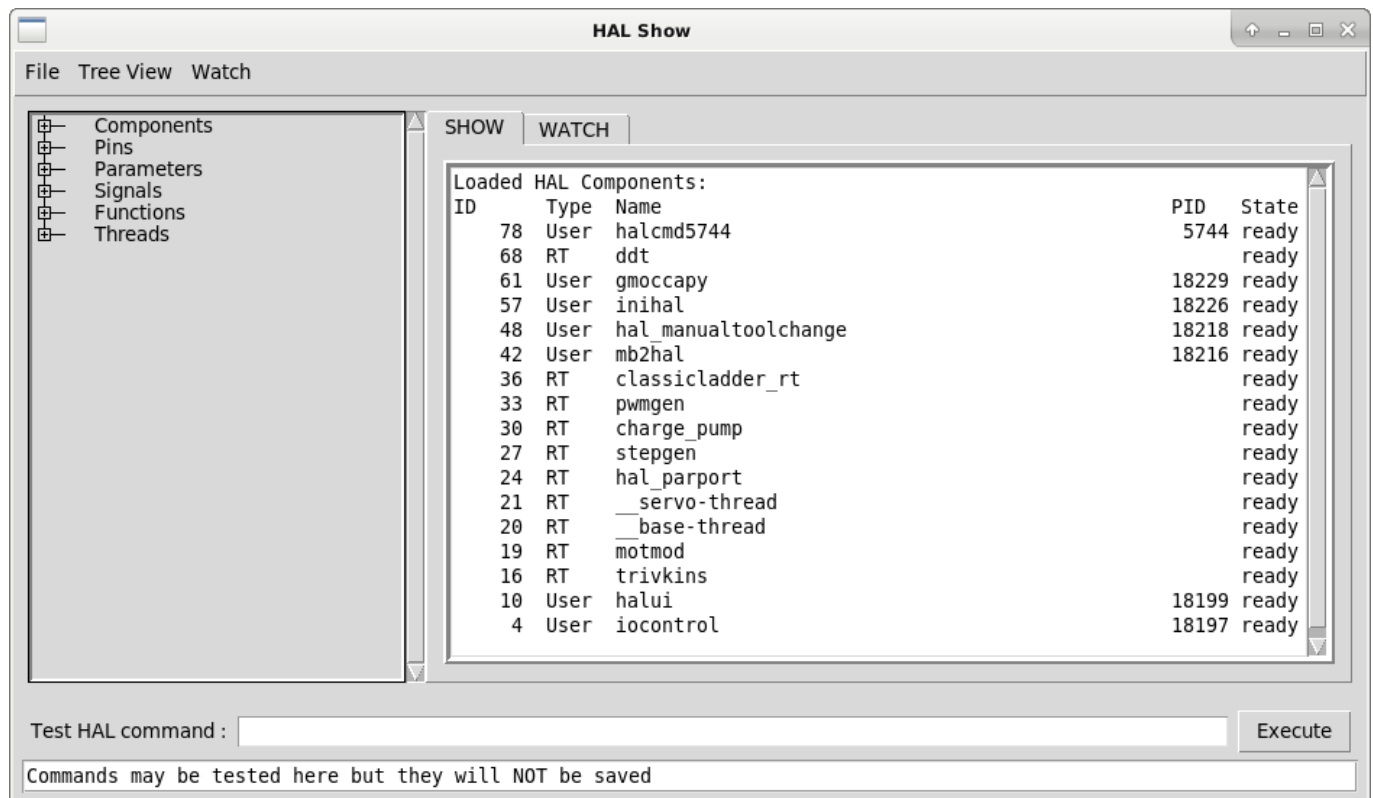


Figure 15.24: Halshow Layout

The tree shows all of the major parts of a HAL. In front of each is a small plus (+) or minus (-) sign in a box. Clicking the plus will expand that tree node to display what is under it. If that box shows a minus sign, clicking it will collapse that section of the tree.

You can also expand or collapse the tree display using the Tree View menu at the upper left edge of the display. Under Tree View you will find: Expand Tree, Collapse Tree; Expand Pins, Expand Parameters, Expand Signals; and Erase Watch. (Note that Erase Watch erases *all* previously set watches, you cannot erase just one watch.)

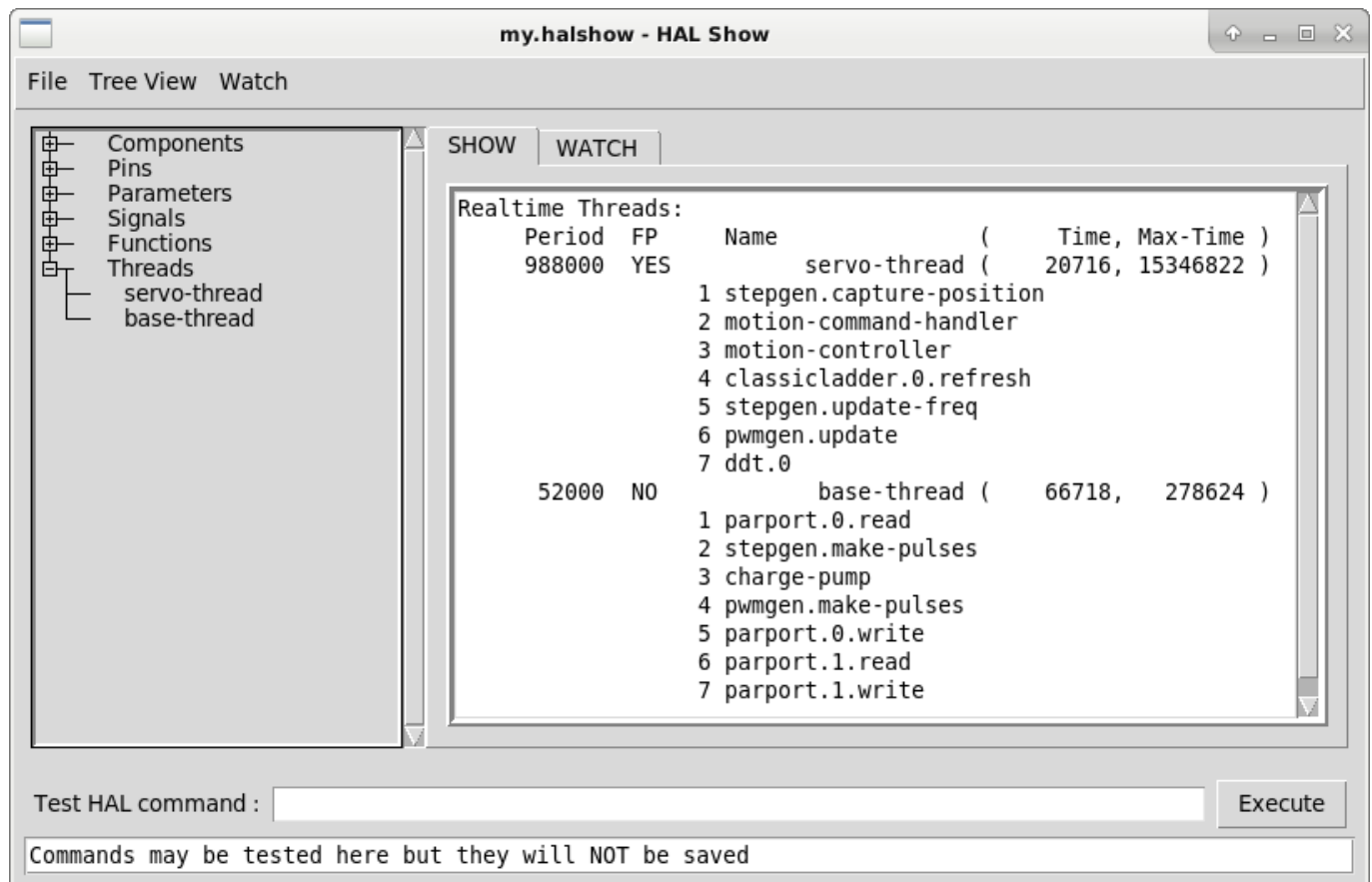


Figure 15.25: Halshow: Show Tab

15.9.3 HAL Show Area

Clicking on the node name, the word “Components” for example, will show you (under the “Show” tab) all that HAL knows about the contents of that node. Figure 15.24 shows a list exactly like you will see if you click the “Components” name. The information display is exactly like those shown in traditional text based HAL analysis tools. The advantage here is that we have mouse click access, access that can be as broad or as focused as you need.

New in 2.9

- Selected text from the Show tab can be copied by right-click or CTRL-C.
- The command entry provides now a BASH-like history (only for the session) for executed commands.

If we take a closer look at the tree display we can see that the six major parts of a HAL can all be expanded at least one level. As these levels are expanded you can get more focused with the reply when you click on the rightmost tree node. You will find that there are some HAL pins and parameters that show more than one reply. This is due to the nature of the search routines in halcmd itself. If you search one pin you may get two, like this:

```
Component Pins:
Owner  Type  Dir  Value  Name
06     bit   -W   TRUE   parport.0.pin-10-in
06     bit   -W   FALSE  parport.0.pin-10-in-not
```

The second pin's name contains the complete name of the first.

Below the show area on the right is a set of widgets that will allow you to play with the running HAL. The commands you enter here and the effect that they have on the running HAL are not saved. They will persist as long as LinuxCNC remains up but are gone as soon as LinuxCNC is.

The entry box labeled "Test HAL Command:" will accept any of the commands listed for halcmd. These include:

- loadrt, unloadrt (load/unload real-time module)
- loadusr, unloadusr (load/unload user-space component)
- addf, delf (add/delete a function to/from a real-time thread)
- net (create a connection between two or more items)
- setp (set parameter (or pin) to a value)

This little editor will enter a command any time you press <enter> or push the execute button. An error message from halcmd will show below this entry widget when these commands are not properly formed. If you are not certain how to set up a proper command you'll need to read again the documentation on halcmd and the specific modules that you are working with.

Let's use this editor to add a differential module to a HAL and connect it to axis position so that we could see the rate of change in position, i.e., acceleration. We first need to load a HAL component named ddt, add it to the servo thread, then connect it to the position pin of a joint. Once that is done we can find the output of the differentiator in halscope. So let's go.

```
loadrt ddt
```

Now look at the components node and you should see ddt in there someplace.

Loaded HAL Components:

ID	Type	Name
10	User	halcmd29800
09	User	halcmd29374
08	RT	ddt
06	RT	hal_parport
05	RT	scope_rt
04	RT	stepgen
03	RT	motmod
02	User	iocontrol

Sure enough there it is. Notice that its ID is 08. Next we need to find out what functions are available with it so we look at functions:

Exported Functions:

Owner	CodeAddr	Arg	FP	Users	Name
08	E0B97630	E0DC7674	YES	0	ddt.0
03	E0DEF83C	00000000	YES	1	motion-command-handler
03	E0DF0BF3	00000000	YES	1	motion-controller
06	E0B541FE	E0DC75B8	NO	1	parport.0.read
06	E0B54270	E0DC75B8	NO	1	parport.0.write
06	E0B54309	E0DC75B8	NO	0	parport.read-all
06	E0B5433A	E0DC75B8	NO	0	parport.write-all
05	E0AD712D	00000000	NO	0	scope.sample
04	E0B618C1	E0DC7448	YES	1	stepgen.capture-position
04	E0B612F5	E0DC7448	NO	1	stepgen.make-pulses
04	E0B614AD	E0DC7448	YES	1	stepgen.update-freq

Here we look for owner #08 and see a function named ddt.0. We should be able to add ddt.0 to the servo thread and it will do its math each time the servo thread is updated. Once again we look up the addf command and find that it uses three arguments like this:

```
addf <funcname> <threadname> [<position>]
```

We already know the funcname=ddt.0 so let's get the thread name right by expanding the thread node in the tree. Here we see two threads, servo-thread and base-thread. The position of ddt.0 in the thread is not critical. So we add the function ddt.0 to the servo-thread:

```
addf ddt.0 servo-thread
```

This is just for viewing, so we leave position blank and get the last position in the thread. The following figure shows the state of halshow after this command has been issued.

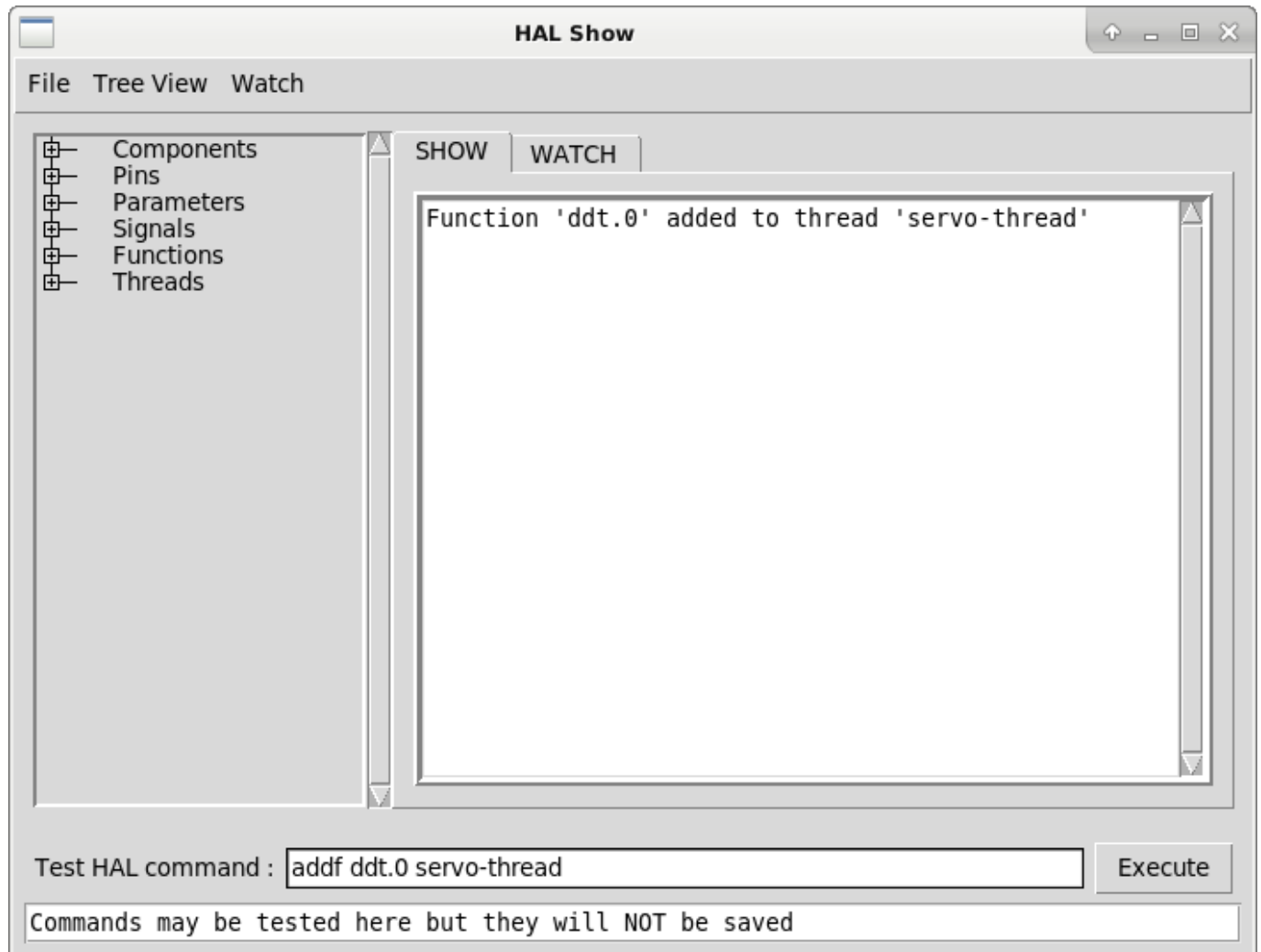


Figure 15.26: Addf command

Next we need to connect ddt to something. But how do we know what pins are available? The answer is to look under pins. There we find ddt and see this:

```
Component Pins:
Owner Type Dir Value Name
08 float R- 0.00000e+00 ddt.0.in
08 float -W 0.00000e+00 ddt.0.out
```

That looks easy enough to understand, but what signal or pin do we want to connect to it? It could be an axis pin, a stepgen pin, or a signal. We see this when we look at joint.0:

```
Component Pins:
Owner Type Dir Value      Name
03    float -W  0.000000e+00 joint.0.motor-pos-cmd ==> Xpos-cmd
```

So it looks like Xpos-cmd should be a good signal to use. Back to the editor where we enter the following command:

```
linksp Xpos-cmd ddt.0.in
```

Now if we look at the Xpos-cmd signal using the tree node we'll see what we've done:

```
Signals:
Type Value Name
float 0.000000e+00 Xpos-cmd
<== joint.0.motor-pos-cmd
==> ddt.0.in
==> stepgen.0.position-cmd
```

We see that this signal comes from joint.o.motor-pos-cmd and goes to both ddt.0.in and stepgen.0.position-cmd. By connecting our block to the signal we have avoided any complications with the normal flow of this motion command.

The HAL Show Area uses halcmd to discover what is happening in a running HAL. It gives you complete information about what it has discovered. It also updates as you issue commands from the little editor panel to modify that HAL. There are times when you want a different set of things displayed without all of the information available in this area. That is where the HAL Watch Area is of value.

15.9.4 Watch Tab Area

New in 2.9

- Buttons for pin/signal/parameter manipulation
- Right-click menu to
 - Copy name
 - Set value
 - Unlink pin
 - Remove from view
- Menu entries for
 - Add signals/pins/parameters by name
 - Set watch interval

Clicking the watch tab produces a blank canvas. You can add signals and pins to this canvas and watch their values.⁷ You can add signals or pins when the watch tab is displayed by clicking on the name of it in the tree view. The following figure shows this canvas with several pins. The pins and signals that are writable have buttons for manipulation on the right side. Pins that are linked to a signal have disabled buttons. To set these values, the corresponding pin has to be unlinked from the signal. That can be done by right-click on the signal name and select "Unlink pin", see Figure 15.28.

⁷The refresh rate of the watch display is much lower than Halmeter or Halscope. If you need good resolution of the timing of signals use these tools or set the interval in the Watch menu.



Figure 15.27: Halshow: Watch Tab

Watch displays bit type (binary) values using colored circles representing LEDs. They show as dark red when a bit signal or pin is false, and as light yellow whenever that signal is true. If you select a pin or signal that is not a bit type (binary) signal, watch will show it as a numerical value.

Watch will quickly allow you to test switches or see the effect of changes that you make to LinuxCNC while using the graphical interface. Watch's refresh rate is a bit slow to see stepper pulses, but you can use it for these if you move an axis very slowly or in very small increments of distance. If you've used IO_Show in LinuxCNC, the watch page in halshow can be set up to watch a parport much as IO_Show did.

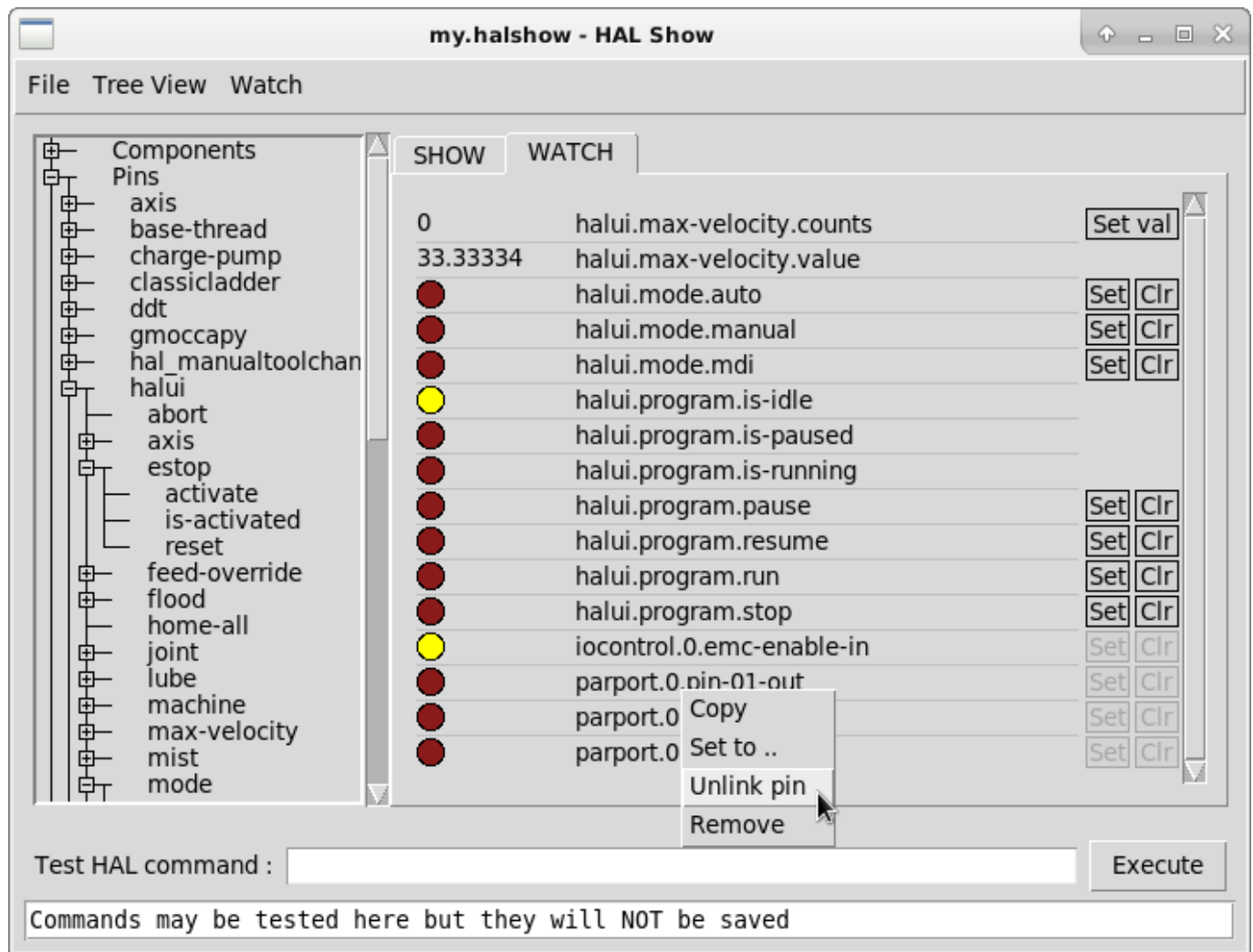


Figure 15.28: Halshow: Watch Tab Context Menu

15.10 HAL Component List

15.10.1 Commands and Userspace Components

All of the commands in the following list have man pages. Some will have expanded descriptions, some will have limited descriptions. Also, all of the components listed below have man pages. From this list you know what components exist, and you can use *man n name* to get additional information. To view the information in the man page, in a terminal window type:

```
man axis (or perhaps 'man 1 axis' if your system requires it.)
```

axis

AXIS LinuxCNC (The Enhanced Machine Controller) Graphical User Interface.

axis-remote

AXIS Remote Interface.

comp

Build, compile and install LinuxCNC HAL components.

gladevcp

Virtual Control Panel for LinuxCNC based on Glade, Gtk and HAL widgets.

gs2

HAL userspace component for Automation Direct GS2 VFD's.

halcmd

Manipulate the Enhanced Machine Controller HAL from the command line.

hal_input

Control HAL pins with any Linux input device, including USB HID devices.

halmeter

Observe HAL pins, signals, and parameters.

halrun

Manipulate the Enhanced Machine Controller HAL from the command line.

halsampler

Sample data from HAL in realtime.

halstreamer

Stream file data into HAL in real time.

halui

Observe HAL pins and command LinuxCNC through NML.

io

Accepts NML I/O commands, interacts with HAL in userspace.

iocontrol

Accepts NML I/O commands, interacts with HAL in userspace.

linuxcnc

LinuxCNC (The Enhanced Machine Controller).

pyvcp

Virtual Control Panel for LinuxCNC.

shuttle

control HAL pins with the ShuttleXpress and ShuttlePRO devices made by Contour Design.

15.10.2 Realtime Components List

Some of these will have expanded descriptions from the man pages. Some will have limited descriptions. All of the components have man pages. From this list you know what components exist and can use *man n name* to get additional information. To view the information in the man page, in a terminal window type:

```
man motion
# or
man 9 motion
```

Note

If component requires a thread with a float, it is usually the slowest, thus *servo-thread*.

Note

See also the *Man Pages* section of the [docs main page](#) or the [directory listing of man9](#).

15.10.2.1 Core LinuxCNC components

This lists HAL components that are all written in lower case and should not be confused with other parts of LinuxCNC bearing the same name (AXIS, ClassicLadder, GladeVCP) that these components communicate with.

motion

Accepts NML motion commands, interacts with HAL in realtime.

axis

Accepts NML motion commands, interacts with HAL in realtime.

classicladder

Realtime software PLC based on ladder logic. See [ClassicLadder](#) chapter for more information.

gladevcp

Displays Virtual Control Panels built with GTK/Glade.

threads

Creates hard realtime HAL threads.

15.10.2.2 Logic and Bitwise components

and2

Two-input AND gate. For out to be true both inputs must be true. [Details](#)

not

Inverter. [Details](#)

or2

Two-input OR gate. [Details](#)

xor2

Two-input XOR (exclusive OR) gate. [Details](#)

dbounce

Filter noisy digital inputs. [Details](#).

debounce

Filter noisy digital inputs. [Details](#). [Description](#)

edge

Edge detector.

flipflop

D type flip-flop.

oneshot

One-shot pulse generator.

logic

General logic function component.

lut5

A 5-input logic function based on a look-up table. [Description](#)

match8

8-bit binary match detector.

select8

8-bit binary match detector.

15.10.2.3 Arithmetic and float-components

abs

Compute the absolute value and sign of the input signal.

blend

Perform linear interpolation between two values.

comp

Two input comparator with hysteresis.

constant

Use a parameter to set the value of a pin.

sum2

Sum of two inputs (each with a gain) and an offset.

counter

Counts input pulses (deprecated).

Use the *encoder* component with ... *counter-mode* = *TRUE*. See section [encoder](#).

updown

Counts up or down, with optional limits and wraparound behavior.

ddt

Compute the derivative of the input function.

deadzone

Return the center if within the threshold.

hypot

Three-input hypotenuse (Euclidean distance) calculator.

mult2

Product of two inputs (multiplexing).

mux16

Select from one of sixteen input values (multiplexing).

mux2

Select from one of two input values (multiplexing).

mux4

Select from one of four input values (multiplexing).

mux8

Select from one of eight input values (multiplexing).

near

Determine whether two values are roughly equal.

offset

Adds an offset to an input, and subtracts it from the feedback value.

integ

Integrator.

invert

Compute the inverse of the input signal.

wcomp

Window comparator.

weighted_sum

Convert a group of bits to an integer.

biquad

Biquad IIR filter

lowpass

Low-pass filter

limit1

Limit the output signal to fall between min and max. ⁸

limit2

Limit the output signal to fall between min and max. Limit its slew rate to less than maxv per second. ⁹

limit3

Limit the output signal to fall between min and max. Limit its slew rate to less than maxv per second. Limit its second derivative to less than MaxA per second squared. ¹⁰

maj3

Compute the majority of 3 inputs.

scale

Applies a scale and offset to its input.

15.10.2.4 Type conversion**conv_bit_s32**

Convert a value from bit to s32.

conv_bit_u32

Convert a value from bit to u32.

conv_float_s32

Convert a value from float to s32.

conv_float_u32

Convert a value from float to u32.

conv_s32_bit

Convert a value from s32 to bit.

conv_s32_float

Convert a value from s32 to float.

conv_s32_u32

Convert a value from s32 to u32.

conv_u32_bit

Convert a value from u32 to bit.

conv_u32_float

Convert a value from u32 to float.

conv_u32_s32

Convert a value from u32 to s32.

⁸When the input is a position, this means that the *position* is limited.

⁹When the input is a position, this means that *position* and *velocity* are limited.

¹⁰When the input is a position, this means that the *position*, *velocity*, and *acceleration* are limited.

15.10.2.5 Hardware Drivers

hal_ppmc

Pico Systems [driver](#) for analog servo, PWM and Stepper controller.

hm2_7i43

Mesa Electronics driver for the 7i43 EPP Anything IO board with HostMot2. (See the man page for more information)

hm2_pci

Mesa Electronics driver for the 5i20, 5i22, 5i23, 4i65, and 4i68 Anything I/O boards, with HostMot2 firmware. (See the man page for more information)

hostmot2

Mesa Electronics [driver](#) for the HostMot2 firmware.

mesa_7i65

Mesa Electronics driver for the 7i65 eight-axis servo card. (See the man page for more information)

pluto_servo

Pluto-P [driver](#) and firmware for the parallel port FPGA, for servos.

pluto_step

Pluto-P [driver](#) for the parallel port FPGA, for steppers.

thc

Torch Height Control using a Mesa THC card or any analog to velocity input

serport

Hardware driver for the digital I/O bits of the 8250 and 16550 serial port.

15.10.2.6 Kinematics

kins

kinematics definitions for LinuxCNC.

gantrykins

A kinematics module that maps one axis to multiple joints.

genhexkins

Gives six degrees of freedom in position and orientation (XYZABC). The location of the motors is defined at compile time.

genserkins

Kinematics that can model a general serial-link manipulator with up to 6 angular joints.

maxkins

Kinematics for a tabletop 5 axis mill named *max* with tilting head (B axis) and horizontal rotary mounted to the table (C axis). Provides UVW motion in the rotated coordinate system. The source file, maxkins.c, may be a useful starting point for other 5-axis systems.

tripodkins

The joints represent the distance of the controlled point from three predefined locations (the motors), giving three degrees of freedom in position (XYZ).

trivkins

There is a 1:1 correspondence between joints and axes. Most standard milling machines and lathes use the trivial kinematics module.

pumakins

Kinematics for PUMA-style robots.

rotatekins

The X and Y axes are rotated 45 degrees compared to the joints 0 and 1.

scarakins

Kinematics for SCARA-type robots.

15.10.2.7 Motor control**at_pid**

Proportional/integral/derivative controller with auto tuning.

pid

Proportional/integral/derivative controller. [Description](#)

pwmgen

Software PWM/PDM generation. [Description](#)

encoder

Software counting of quadrature encoder signals. [Description](#).

stepgen

Software step pulse generation. [Description](#).

15.10.2.8 BLDC and 3-phase motor control**bldc_hall3**

3-wire Bipolar trapezoidal commutation BLDC motor driver using Hall sensors.

clarke2

Two input version of Clarke transform.

clarke3

Clarke (3 phase to cartesian) transform.

clarkeinv

Inverse Clarke transform.

15.10.2.9 Other Components**charge_pump**

Creates a square-wave for the *charge pump* input of some controller boards. The *Charge Pump* should be added to the base thread function. When enabled the output is on for one period and off for one period. To calculate the frequency of the output $1/(\text{period time in seconds} \times 2) = \text{Hz}$. For example if you have a base period of 100,000 ns that is 0.0001 seconds and the formula would be $1/(0.0001 \times 2) = 5,000 \text{ Hz}$ or 5 kHz.

encoder_ratio

An electronic gear to synchronize two axes.

estop_latch

ESTOP latch.

feedcomp

Multiply the input by the ratio of current velocity to the feed rate.

gearchange

Select from one of two speed ranges.

ilowpass

While it may find other applications, this component was written to create smoother motion while jogging with an MPG.

In a machine with high acceleration, a short jog can behave almost like a step function. By putting the ilowpass component between the MPG encoder counts output and the axis jog-counts input, this can be smoothed.

Choose scale conservatively so that during a single session there will never be more than about $2e9/\text{scale}$ pulses seen on the MPG. Choose gain according to the smoothing level desired. Divide the axis.N.jog-scale values by scale.

joyhandle

Sets nonlinear joystick movements, deadbands and scales.

knob2float

Convert counts (probably from an encoder) to a float value.

minmax

Track the minimum and maximum values of the input to the outputs.

sample_hold

Sample and Hold.

sampler

Sample data from HAL in real time.

siggen

Signal generator. [Description](#).

sim_encoder

Simulated quadrature encoder. [Description](#).

sphereprobe

Probe a pretend hemisphere.

steptest

Used by Stepconf to allow testing of acceleration and velocity values for an axis.

streamer

Stream file data into HAL in real time.

supply

Set output pins with values from parameters (deprecated).

threadtest

Component for testing thread behavior.

time

Accumulated run-time timer counts HH:MM:SS of *active* input.

timedelay

The equivalent of a time-delay relay.

timedelta

Component that measures thread scheduling timing behavior.

toggle2nist

Toggle button to nist logic.

toggle

Push-on, push-off from momentary pushbuttons.

tristate_bit

Place a signal on an I/O pin only when enabled, similar to a tristate buffer in electronics.

tristate_float

Place a signal on an I/O pin only when enabled, similar to a tristate buffer in electronics.

watchdog

Monitor one to thirty-two inputs for a *heartbeat*.

15.10.3 HAL API calls

```
hal_add_funct_to_thread.3hal
hal_bit_t.3hal
hal_create_thread.3hal
hal_del_funct_from_thread.3hal
hal_exit.3hal
hal_export_funct.3hal
hal_float_t.3hal
hal_get_lock.3hal
hal_init.3hal
hal_link.3hal
hal_malloc.3hal
hal_param_bit_new.3hal
hal_param_bit_newf.3hal
hal_param_float_new.3hal
hal_param_float_newf.3hal
hal_param_new.3hal
hal_param_s32_new.3hal
hal_param_s32_newf.3hal
hal_param_u32_new.3hal
hal_param_u32_newf.3hal
hal_parport.3hal
hal_pin_bit_new.3hal
hal_pin_bit_newf.3hal
hal_pin_float_new.3hal
hal_pin_float_newf.3hal
hal_pin_new.3hal
hal_pin_s32_new.3hal
hal_pin_s32_newf.3hal
hal_pin_u32_new.3hal
hal_pin_u32_newf.3hal
hal_ready.3hal
hal_s32_t.3hal
hal_set_constructor.3hal
hal_set_lock.3hal
hal_signal_delete.3hal
hal_signal_new.3hal
hal_start_threads.3hal
hal_type_t.3hal
hal_u32_t.3hal
hal_unlink.3hal
intro.3hal
undocumented.3hal
```

15.10.4 RTAPI calls

```
EXPORT_FUNCTION.3rtapi
MODULE_AUTHOR.3rtapi
MODULE_DESCRIPTION.3rtapi
MODULE_LICENSE.3rtapi
RTAPI_MP_ARRAY_INT.3rtapi
RTAPI_MP_ARRAY_LONG.3rtapi
RTAPI_MP_ARRAY_STRING.3rtapi
RTAPI_MP_INT.3rtapi
RTAPI_MP_LONG.3rtapi
RTAPI_MP_STRING.3rtapi
intro.3rtapi
rtapi_app_exit.3rtapi
rtapi_app_main.3rtapi
rtapi_clock_set_period.3rtapi
rtapi_delay.3rtapi
rtapi_delay_max.3rtapi
rtapi_exit.3rtapi
rtapi_get_clocks.3rtapi
rtapi_get_msg_level.3rtapi
rtapi_get_time.3rtapi
rtapi_inb.3rtapi
rtapi_init.3rtapi
rtapi_module_param.3rtapi
RTAPI_MP_ARRAY_INT.3rtapi
RTAPI_MP_ARRAY_LONG.3rtapi
RTAPI_MP_ARRAY_STRING.3rtapi
RTAPI_MP_INT.3rtapi
RTAPI_MP_LONG.3rtapi
RTAPI_MP_STRING.3rtapi
rtapi_mutex.3rtapi
rtapi_outb.3rtapi
rtapi_print.3rtapi
rtapi_prio.3rtapi
rtapi_prio_highest.3rtapi
rtapi_prio_lowest.3rtapi
rtapi_prio_next_higher.3rtapi
rtapi_prio_next_lower.3rtapi
rtapi_region.3rtapi
rtapi_release_region.3rtapi
rtapi_request_region.3rtapi
rtapi_set_msg_level.3rtapi
rtapi_shmem.3rtapi
rtapi_shmem_delete.3rtapi
rtapi_shmem_getptr.3rtapi
rtapi_shmem_new.3rtapi
rtapi_snprintf.3rtapi
rtapi_task_delete.3rtapi
rtapi_task_new.3rtapi
rtapi_task_pause.3rtapi
rtapi_task_resume.3rtapi
rtapi_task_start.3rtapi
rtapi_task_wait.3rtapi
```


15.11 HAL Component Descriptions

15.11.1 Stepgen

This component provides software based generation of step pulses in response to position or velocity commands. In position mode, it has a built in pre-tuned position loop, so PID tuning is not required. In velocity mode, it drives a motor at the commanded speed, while obeying velocity and acceleration limits. It is a realtime component only, and depending on CPU speed, etc, is capable of maximum step rates of 10 kHz to perhaps 50 kHz. The step pulse generator block diagram shows three block diagrams, each is a single step pulse generator. The first diagram is for step type 0, (step and direction). The second is for step type 1 (up/down, or pseudo-PWM), and the third is for step types 2 through 14 (various stepping patterns). The first two diagrams show position mode control, and the third one shows velocity mode. Control mode and step type are set independently, and any combination can be selected.

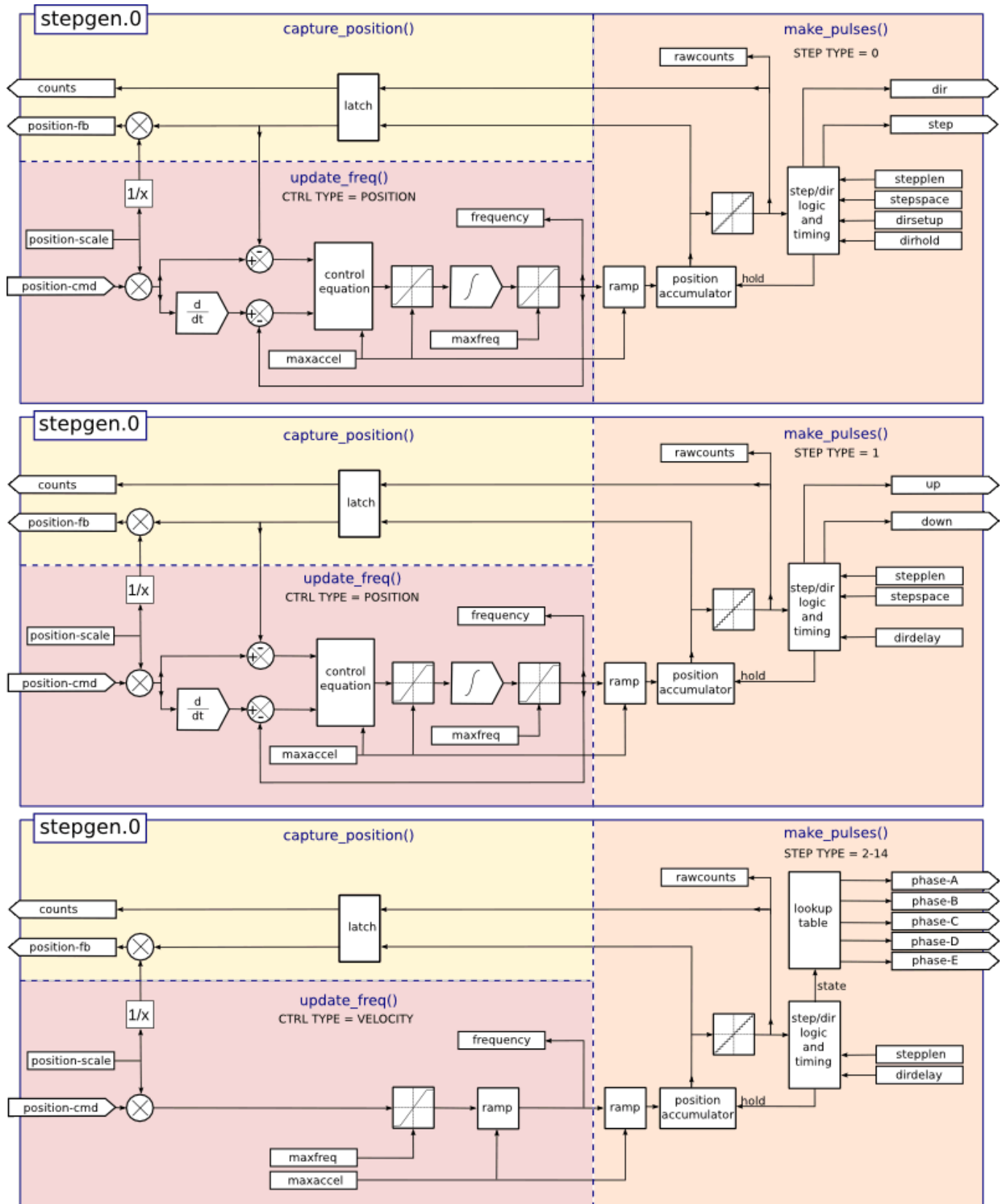


Figure 15.29: Step Pulse Generator Block Diagram position mode

Loading stepgen component

```
halcmd: loadrt stepgen step_type=<type-array> [ctrl_type=<ctrl_array>]
```

<type-array>

is a series of comma separated decimal integers. Each number causes a single step pulse generator to be loaded, the value of the number determines the stepping type.

<ctrl_array>

is a comma separated series of *p* or *v* characters, to specify position or velocity mode.

ctrl_type

is optional, if omitted, all of the step generators will be position mode.

For example:

```
halcmd: loadrt stepgen step_type=0,0,2 ctrl_type=p,p,v
```

Will install three step generators. The first two use step type 0 (step and direction) and run in position mode. The last one uses step type 2 (quadrature) and runs in velocity mode. The default value for *<config-array>* is 0,0,0 which will install three type 0 (step/dir) generators. The maximum number of step generators is 8 (as defined by MAX_CHAN in stepgen.c). Each generator is independent, but all are updated by the same function(s) at the same time. In the following descriptions, *<chan>* is the number of a specific generator. The first generator is number 0.

Unloading stepgen component

```
halcmd: unloadrt stepgen
```

15.11.1.1 Pins

On the step type and control type selected.

- (float) *stepgen.<chan>.position-cmd* - Desired motor position, in position units (position mode only).
- (float) *stepgen.<chan>.velocity-cmd* - Desired motor velocity, in position units per second (velocity mode only).
- (s32) *stepgen.<chan>.counts* - Feedback position in counts, updated by *capture_position()*.
- (float) *stepgen.<chan>.position-fb* - Feedback position in position units, updated by *capture_position()*.
- (bit) *stepgen.<chan>.enable* - Enables output steps - when false, no steps are generated.
- (bit) *stepgen.<chan>.step* - Step pulse output (step type 0 only).
- (bit) *stepgen.<chan>.dir* - Direction output (step type 0 only).
- (bit) *stepgen.<chan>.up* - UP pseudo-PWM output (step type 1 only).
- (bit) *stepgen.<chan>.down* - DOWN pseudo-PWM output (step type 1 only).
- (bit) *stepgen.<chan>.phase-A* - Phase A output (step types 2-14 only).
- (bit) *stepgen.<chan>.phase-B* - Phase B output (step types 2-14 only).
- (bit) *stepgen.<chan>.phase-C* - Phase C output (step types 3-14 only).
- (bit) *stepgen.<chan>.phase-D* - Phase D output (step types 5-14 only).
- (bit) *stepgen.<chan>.phase-E* - Phase E output (step types 11-14 only).

15.11.1.2 Parameters

- (float) *stepgen.<chan>.position-scale* - Steps per position unit. This parameter is used for both output and feedback.
- (float) *stepgen.<chan>.maxvel* - Maximum velocity, in position units per second. If 0.0, has no effect.
- (float) *stepgen.<chan>.maxaccel* - Maximum accel/decel rate, in positions units per second squared. If 0.0, has no effect.
- (float) *stepgen.<chan>.frequency* - The current step rate, in steps per second.
- (float) *stepgen.<chan>.steplen* - Length of a step pulse (step type 0 and 1) or minimum time in a given state (step types 2-14), in nano-seconds.
- (float) *stepgen.<chan>.stepspace* - Minimum spacing between two step pulses (step types 0 and 1 only), in nano-seconds. Set to 0 to enable the *stepgen doublefreq* function. To use *doublefreq* the [parport reset function](#) must be enabled.
- (float) *stepgen.<chan>.dirsetup* - Minimum time from a direction change to the beginning of the next step pulse (step type 0 only), in nanoseconds.
- (float) *stepgen.<chan>.dirhold* - Minimum time from the end of a step pulse to a direction change (step type 0 only), in nanoseconds.
- (float) *stepgen.<chan>.dirdelay* - Minimum time any step to a step in the opposite direction (step types 1-14 only), in nano-seconds.
- (s32) *stepgen.<chan>.rawcounts* - The raw feedback count, updated by *make_pulses()*.

In position mode, the values of *maxvel* and *maxaccel* are used by the internal position loop to avoid generating step pulse trains that the motor cannot follow. When set to values that are appropriate for the motor, even a large instantaneous change in commanded position will result in a smooth trapezoidal move to the new location. The algorithm works by measuring both position error and velocity error, and calculating an acceleration that attempts to reduce both to zero at the same time. For more details, including the contents of the *control equation* box, consult the code.

In velocity mode, *maxvel* is a simple limit that is applied to the commanded velocity, and *maxaccel* is used to ramp the actual frequency if the commanded velocity changes abruptly. As in position mode, proper values for these parameters ensure that the motor can follow the generated pulse train.

15.11.1.3 Step Types

Step generator supports 15 different *step sequences*:

Step Type 0 Step type 0 is the standard step and direction type. When configured for step type 0, there are four extra parameters that determine the exact timing of the step and direction signals. In the following figure the meaning of these parameters is shown. The parameters are in nanoseconds, but will be rounded up to an integer multiple of the thread period for the thread that calls *make_pulses()*. For example, if *make_pulses()* is called every 16 μ s, and *steplen* is 20000, then the step pulses will be $2 \times 16 = 32 \mu$ s long. The default value for all four of the parameters is 1ns, but the automatic rounding takes effect the first time the code runs. Since one step requires *steplen* ns high and *stepspace* ns low, the maximum frequency is 1,000,000,000 divided by (*steplen*+*stepspace*). If *maxfreq* is set higher than that limit, it will be lowered automatically. If *maxfreq* is zero, it will remain zero, but the output frequency will still be limited.

When using the parallel port driver the step frequency can be doubled using the [parport reset](#) function together with *stepgen's doublefreq* setting.

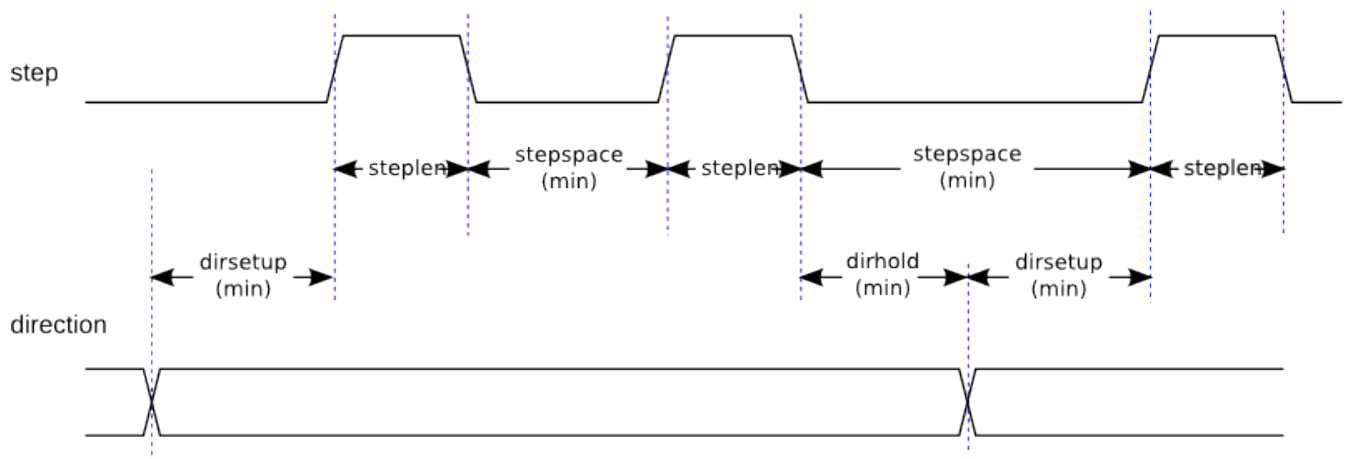


Figure 15.30: Step and Direction Timing

Step Type 1 Step type 1 has two outputs, up and down. Pulses appear on one or the other, depending on the direction of travel. Each pulse is *steplen* ns long, and the pulses are separated by at least *stepspace* ns. The maximum frequency is the same as for step type 0. If *maxfreq* is set higher than the limit it will be lowered. If *maxfreq* is zero, it will remain zero but the output frequency will still be limited.

**Warning**

Do not use the parport reset function with step types 2 - 14. Unexpected results can happen.

Step Type 2 - 14 Step types 2 through 14 are state based, and have from two to five outputs. On each step, a state counter is incremented or decremented. The Two-and-Three-Phase, Four-Phase, and Five-Phase show the output patterns as a function of the state counter. The maximum frequency is 1,000,000,000 divided by *steplen*, and as in the other modes, *maxfreq* will be lowered if it is above the limit.

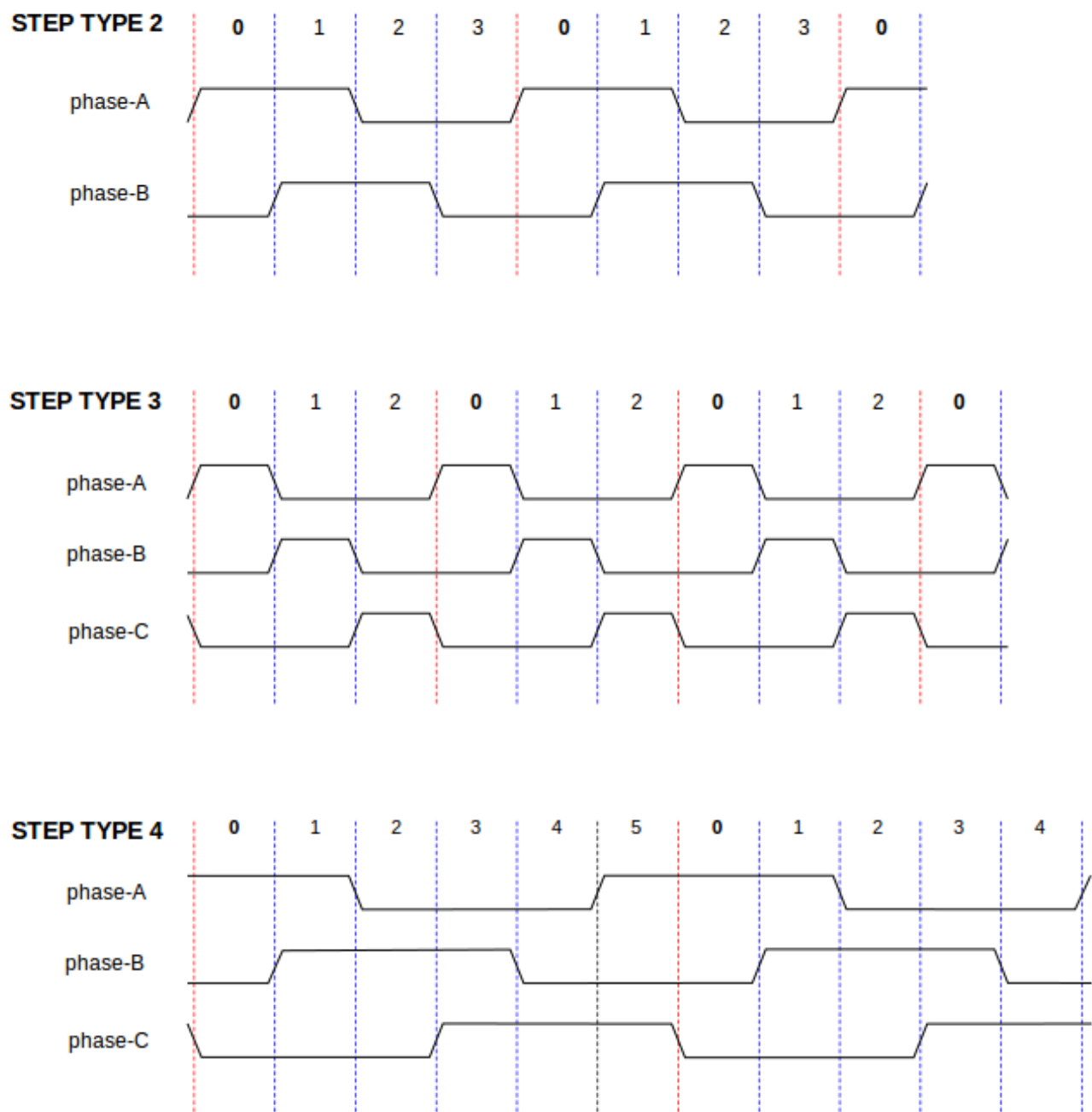


Figure 15.31: Two-and-Three-Phase Step Types

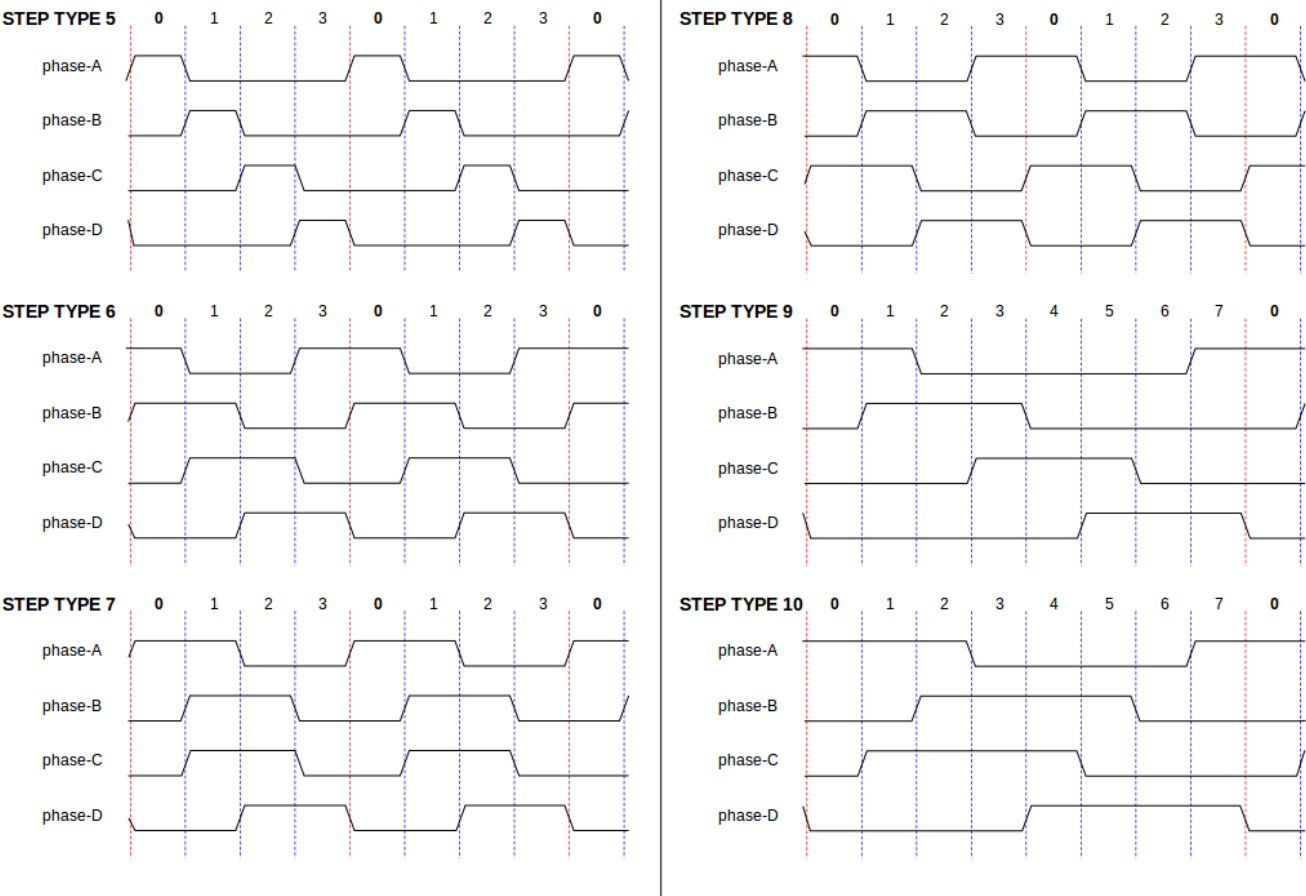


Figure 15.32: Four-Phase Step Types

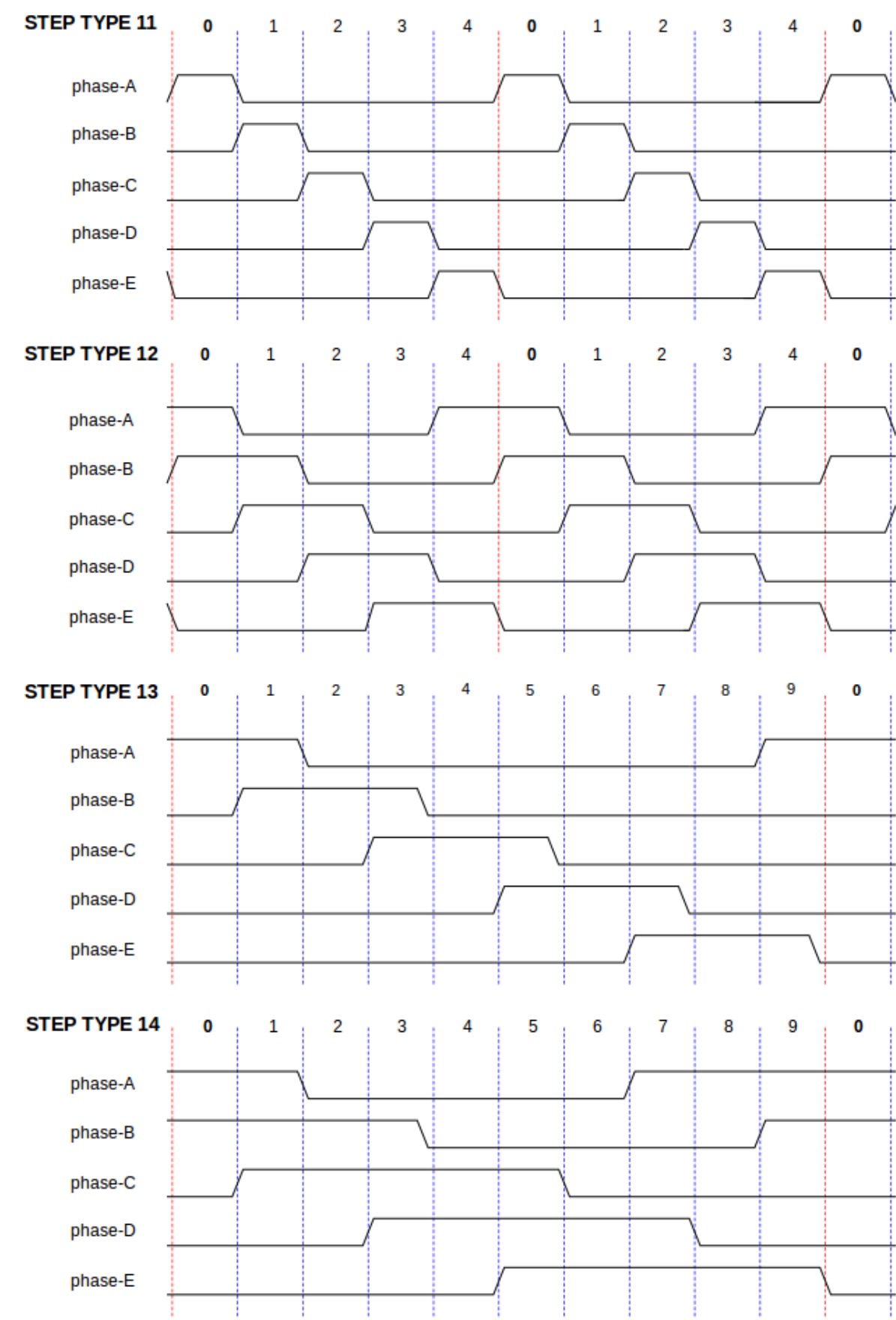


Figure 15.33: Five-Phase Step Types

15.11.1.4 Functions

The component exports three functions. Each function acts on all of the step pulse generators - running different generators in different threads is not supported.

- (funct) *stepgen.make-pulses* - High speed function to generate and count pulses (no floating point).
- (funct) *stepgen.update-freq* - Low speed function does position to velocity conversion, scaling and limiting.
- (funct) *stepgen.capture-position* - Low speed function for feedback, updates latches and scales position.

The high speed function *stepgen.make-pulses* should be run in a very fast thread, from 10 to 50 μ s depending on the capabilities of the computer. That thread's period determines the maximum step frequency, since *steplen*, *stepspace*, *dirsetup*, *dirhold*, and *dirdelay* are all rounded up to a integer multiple of the thread period in nanoseconds. The other two functions can be called at a much lower rate.

15.11.2 PWMgen

This component provides software based generation of PWM (Pulse Width Modulation) and PDM (Pulse Density Modulation) waveforms. It is a realtime component only, and depending on CPU speed, etc, is capable of PWM frequencies from a few hundred Hertz at pretty good resolution, to perhaps 10 kHz with limited resolution.

Loading PWMgen

```
loadrt pwmgen output_type=<config-array>
```

The *<config-array>* is a series of comma separated decimal integers. Each number causes a single PWM generator to be loaded, the value of the number determines the output type. The following example will install three PWM generators. There is no default value, if *<config-array>* is not specified, no PWM generators will be installed. The maximum number of frequency generators is 8 (as defined by MAX_CHAN in pwmgen.c). Each generator is independent, but all are updated by the same function(s) at the same time. In the following descriptions, *<chan>* is the number of a specific generator. The first generator is number 0.

Loading PWMgen Example

```
loadrt pwmgen output_type=0,1,2
```

Will install three PWM generators. The first will use an output of type 0 (PWM only), the next one will use a type 1 output (PWM and direction) and the third will use a type 2 output (UP and DOWN). There is no default value, if *<config-array>* is not not specified, no PWM generator will be installed. The maximum number of frequency generators is 8 (as defined by MAX_CHAN in pwmgen.c). Each generator is independent, but all are updated by the same function(s), at the same time. In the descriptions that follow, *<chan>* is the number of specific generators. The numbering of PWM generators starts at 0.

Unloading PWMgen

```
unloadrt pwmgen
```

15.11.2.1 Output Types

The PWM generator supports three different *output types*.

- *Output type 0* - PWM output pin only. Only positive commands are accepted, negative values are treated as zero (and will be affected by the parameter *min-dc* if it is non-zero).
- *Output type 1* - PWM/PDM and direction pins. Positive and negative inputs will be output as positive and negative PWM. The direction pin is false for positive commands, and true for negative commands. If your control needs positive PWM for both CW and CCW use the [abs](#) component to convert your PWM signal to positive value when a negative input is input.
- *Output type 2* - UP and DOWN pins. For positive commands, the PWM signal appears on the up output, and the down output remains false. For negative commands, the PWM signal appears on the down output, and the up output remains false. Output type 2 is suitable for driving most H-bridges.

15.11.2.2 Pins

Each PWM generator will have the following pins:

- (float) *pwmgen.<chan>.value* - Command value, in arbitrary units. Will be scaled by the *scale* parameter (see below).
- (bit) *pwmgen.<chan>.enable* - Enables or disables the PWM generator outputs.

Each PWM generator will also have some of these pins, depending on the output type selected:

- (bit) *pwmgen.<chan>.pwm* - PWM (or PDM) output, (output types 0 and 1 only).
- (bit) *pwmgen.<chan>.dir* - Direction output (output type 1 only).
- (bit) *pwmgen.<chan>.up* - PWM/PDM output for positive input value (output type 2 only).
- (bit) *pwmgen.<chan>.down* - PWM/PDM output for negative input value (output type 2 only).

15.11.2.3 Parameters

- (float) *pwmgen.<chan>.scale* - Scaling factor to convert *value* from arbitrary units to duty cycle. For example if scale is set to 4000 and the input value passed to the *pwmgen.<chan>.value* is 4000 then it will be 100% duty-cycle (always on). If the value is 2000 then it will be a 50% 25 Hz square wave.
- (float) *pwmgen.<chan>.pwm-freq* - Desired PWM frequency, in Hz. If 0.0, generates PDM instead of PWM. If set higher than internal limits, next call of *update_freq()* will set it to the internal limit. If non-zero, and *dither* is false, next call of *update_freq()* will set it to the nearest integer multiple of the *make_pulses()* function period.
- (bit) *pwmgen.<chan>.dither-pwm* - If true, enables dithering to achieve average PWM frequencies or duty cycles that are unobtainable with pure PWM. If false, both the PWM frequency and the duty cycle will be rounded to values that can be achieved exactly.
- (float) *pwmgen.<chan>.min-dc* - Minimum duty cycle, between 0.0 and 1.0 (duty cycle will go to zero when disabled, regardless of this setting).
- (float) *pwmgen.<chan>.max-dc* - Maximum duty cycle, between 0.0 and 1.0.
- (float) *pwmgen.<chan>.curr-dc* - Current duty cycle - after all limiting and rounding (read only).

15.11.2.4 Functions

The component exports two functions. Each function acts on all of the PWM generators - running different generators in different threads is not supported.

- (funct) *pwmgen.make-pulses* - High speed function to generate PWM waveforms (no floating point). The high speed function *pwmgen.make-pulses* should be run in the base (fastest) thread, from 10 to 50 μ s depending on the capabilities of the computer. That thread's period determines the maximum PWM carrier frequency, as well as the resolution of the PWM or PDM signals. If the base thread is 50,000nS then every 50uS the module decides if it is time to change the state of the output. At 50% duty cycle and 25 Hz PWM frequency this means that the output changes state every $(1 / 25)$ seconds / 50uS * 50% = 400 iterations. This also means that you have a 800 possible duty cycle values (without dithering)
- (funct) *pwmgen.update* - Low speed function to scale and limit value and handle other parameters. This is the function of the module that does the more complicated mathematics to work out how many base-periods the output should be high for, and how many it should be low for.

15.11.3 Encoder

This component provides software based counting of signals from quadrature (or single-pulse) encoders. It is a realtime component only, and depending on CPU speed, latency, etc, is capable of maximum count rates of 10 kHz to perhaps up to 50 kHz.

The base thread should be 1/2 count speed to allow for noise and timing variation. For example if you have a 100 pulse per revolution encoder on the spindle and your maximum RPM is 3000 the maximum base thread should be 25 μ s. A 100 pulse per revolution encoder will have 400 counts. The spindle speed of 3000 RPM = 50 RPS (revolutions per second). $400 * 50 = 20,000$ counts per second or 50 μ s between counts.

The Encoder Counter Block Diagram is a block diagram of one channel of an encoder counter.

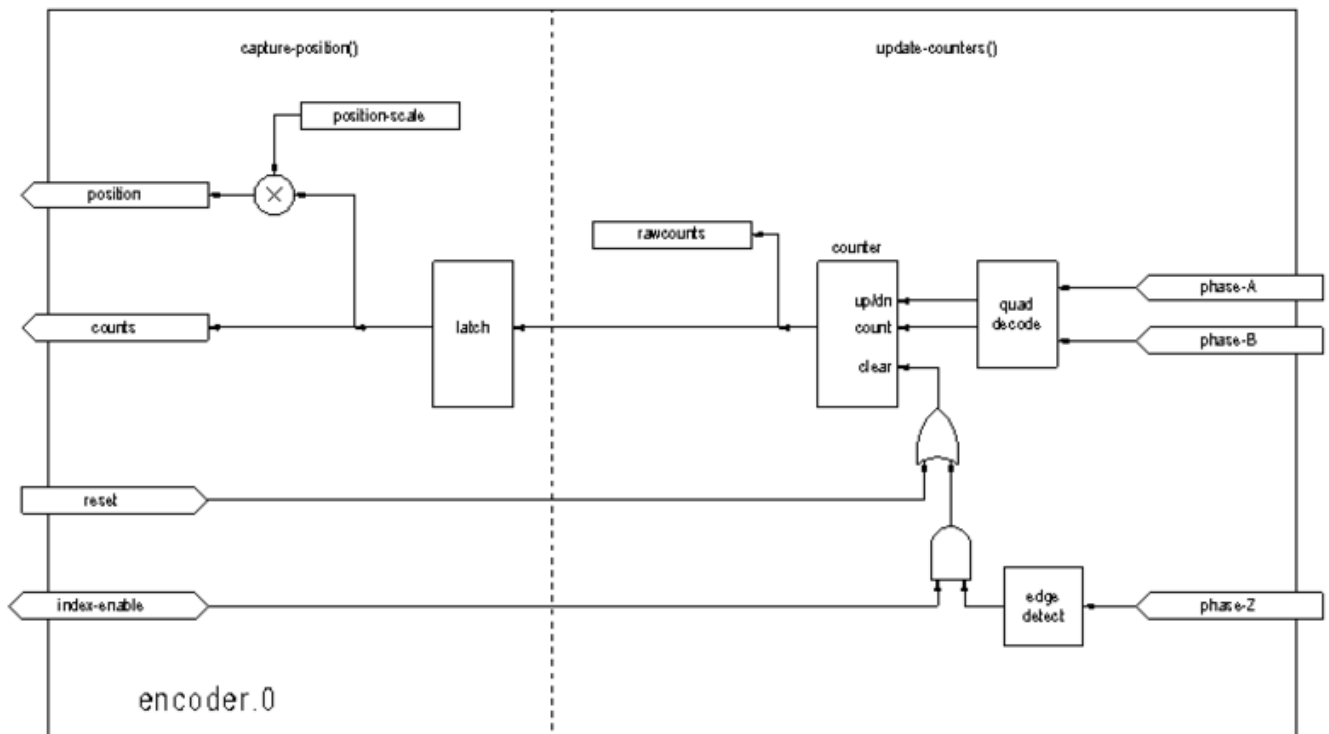


Figure 15.34: Encoder Counter Block Diagram

Loading Encoder

```
halcmd: loadrt encoder [num_chan=<counters>]
```

<counters> is the number of encoder counters that you want to install. If *numchan* is not specified, three counters will be installed. The maximum number of counters is 8 (as defined by `MAX_CHAN` in `encoder.c`). Each counter is independent, but all are updated by the same function(s) at the same time. In the following descriptions, *<chan>* is the number of a specific counter. The first counter is number 0.

Unloading Encoder

```
halcmd: unloadrt encoder
```

15.11.3.1 Pins

- *encoder.<chan>.counter-mode* (bit, I/O) (default: FALSE) - Enables counter mode. When true, the counter counts each rising edge of the phase-A input, ignoring the value on phase-B. This is useful for counting the output of a single channel (non-quadrature) sensor. When false, it counts in quadrature mode.

- *encoder.<chan>.missing-teeth* (s32, In) (default: 0) - Enables the use of missing-tooth index. This allows a single IO pin to provide both position and index information. If the encoder wheel has 58 teeth with two missing, spaced as if there were 60 (common for automotive crank sensors) then the position-scale should be set to 60 and missing-teeth to 2. To use this mode counter-mode should be set true. This mode will work for lathe threading but not for rigid tapping.
- *encoder.<chan>.counts* (s32, Out) - Position in encoder counts.
- *encoder.<chan>.counts-latched* (s32, Out) - Not used at this time.
- *encoder.<chan>.index-enable* (bit, I/O) - When True, *counts* and *position* are reset to zero on next rising edge of Phase Z.
At the same time, *index-enable* is reset to zero to indicate that the rising edge has occurred. The *index-enable* pin is bi-directional. If *index-enable* is False, the Phase Z channel of the encoder will be ignored, and the counter will count normally. The encoder driver will never set *index-enable* True. However, some other component may do so.
- *encoder.<chan>.latch-falling* (bit, In) (default: TRUE) - Not used at this time.
- *encoder.<chan>.latch-input* (bit, In) (default: TRUE) - Not used at this time.
- *encoder.<chan>.latch-rising* (bit, In) - Not used at this time.
- *encoder.<chan>.min-speed-estimate* (float, in) - Determine the minimum true velocity magnitude at which velocity will be estimated as nonzero and position-interpolated will be interpolated. The units of *min-speed-estimate* are the same as the units of *velocity*. Scale factor, in counts per length unit. Setting this parameter too low will cause it to take a long time for velocity to go to 0 after encoder pulses have stopped arriving.
- *encoder.<chan>.phase-A* (bit, In) - Phase A of the quadrature encoder signal.
- *encoder.<chan>.phase-B* (bit, In) - Phase B of the quadrature encoder signal.
- *encoder.<chan>.phase-Z* (bit, In) - Phase Z (index pulse) of the quadrature encoder signal.
- *encoder.<chan>.position* (float, Out) - Position in scaled units (see *position-scale*).
- *encoder.<chan>.position-interpolated* (float, Out) - Position in scaled units, interpolated between encoder counts.
The *position-interpolated* attempts to interpolate between encoder counts, based on the most recently measured velocity. Only valid when velocity is approximately constant and above *min-speed-estimate*. Do not use for position control, since its value is incorrect at low speeds, during direction reversals, and during speed changes.
However, it allows a low ppr encoder (including a one pulse per revolution *encoder*) to be used for lathe threading, and may have other uses as well.
- *encoder.<chan>.position-latched* (float, Out) - Not used at this time.
- *encoder.<chan>.position-scale* (float, I/O) - Scale factor, in counts per length unit. For example, if position-scale is 500, then 1000 counts of the encoder will be reported as a position of 2.0 units.
- *encoder.<chan>.rawcounts* (s32, In) - The raw count, as determined by update-counters. This value is updated more frequently than counts and position. It is also unaffected by reset or the index pulse.
- *encoder.<chan>.reset* (bit, In) - When True, force *counts* and *position* to zero immediately.
- *encoder.<chan>.velocity* (float, Out) - Velocity in scaled units per second. *encoder* uses an algorithm that greatly reduces quantization noise as compared to simply differentiating the *position* output. When the magnitude of the true velocity is below min-speed-estimate, the velocity output is 0.
- *encoder.<chan>.x4-mode* (bit, I/O) (default: TRUE) - Enables times-4 mode. When true, the counter counts each edge of the quadrature waveform (four counts per full cycle). When false, it only counts once per full cycle. In counter-mode, this parameter is ignored. The 1x mode is useful for some jogwheels.

15.11.3.2 Parameters

- *encoder.<chan>.capture-position.time* (s32, RO)
- *encoder.<chan>.capture-position.tmax* (s32, RW)
- *encoder.<chan>.update-counters.time* (s32, RO)
- *encoder.<chan>.update-counter.tmax* (s32, RW)

15.11.3.3 Functions

The component exports two functions. Each function acts on all of the encoder counters - running different counters in different threads is not supported.

- (funct) *encoder.update-counters* - High speed function to count pulses (no floating point).
- (funct) *encoder.capture-position* - Low speed function to update latches and scale position.

15.11.4 PID

This component provides Proportional/Integral/Derivative control loops. It is a realtime component only. For simplicity, this discussion assumes that we are talking about position loops, however this component can be used to implement other feedback loops such as speed, torch height, temperature, etc. The PID Loop Block Diagram is a block diagram of a single PID loop.

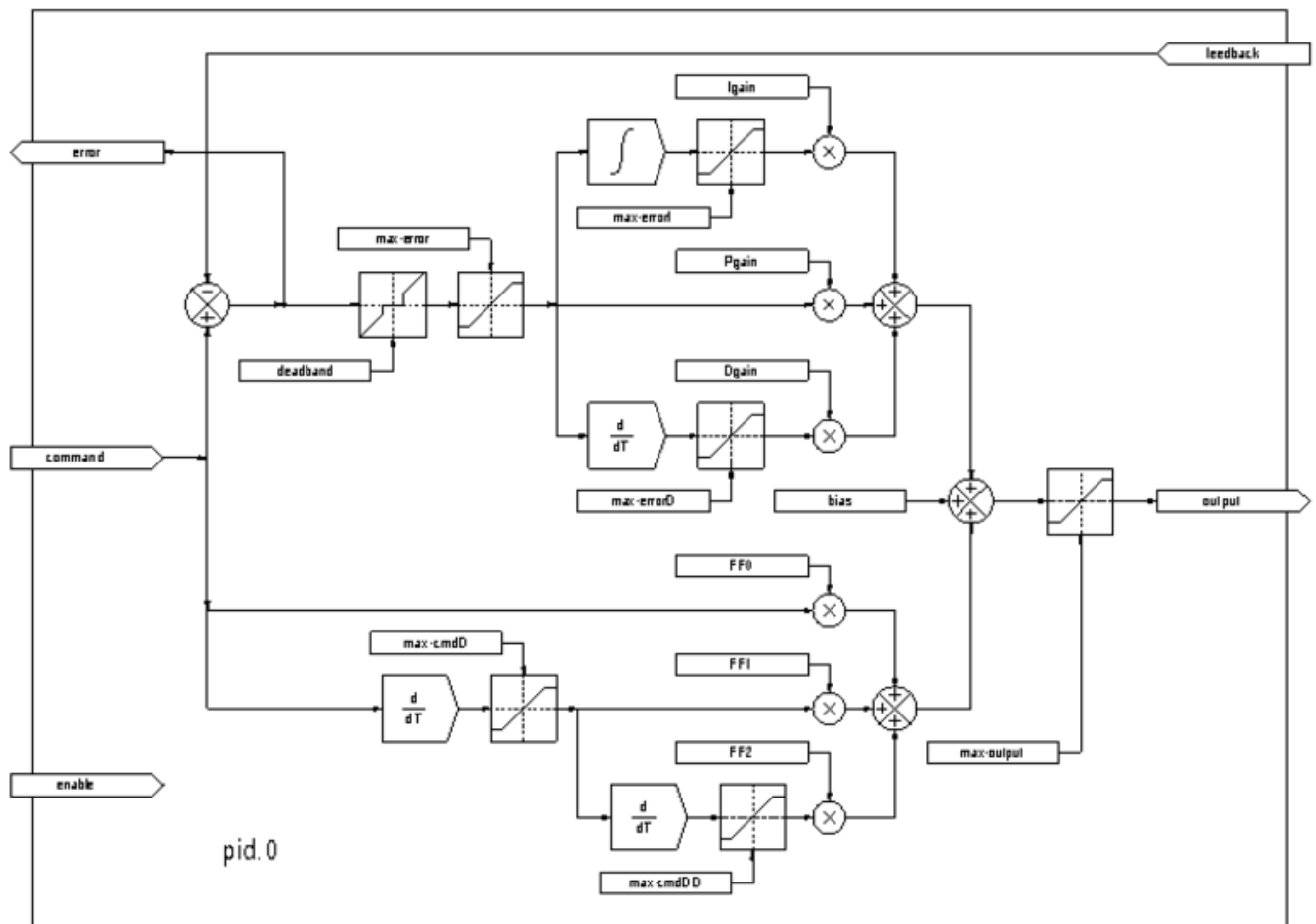


Figure 15.35: PID Loop Block Diagram

Loading PID

```
halcmd: loadrt pid [num_chan=<loops>] [debug=1]
```

<loops> is the number of PID loops that you want to install. If *numchan* is not specified, one loop will be installed. The maximum number of loops is 16 (as defined by MAX_CHAN in pid.c). Each loop is completely independent. In the following descriptions, *<loopnum>* is the loop number of a specific loop. The first loop is number 0.

If *debug=1* is specified, the component will export a few extra pins that may be useful during debugging and tuning. By default, the extra pins are not exported, to save shared memory space and avoid cluttering the pin list.

Unloading PID

```
halcmd: unloadrt pid
```

15.11.4.1 Pins

The three most important pins are

- (float) *pid.<loopnum>.command* - The desired position, as commanded by another system component.
- (float) *pid.<loopnum>.feedback* - The present position, as measured by a feedback device such as an encoder.
- (float) *pid.<loopnum>.output* - A velocity command that attempts to move from the present position to the desired position.

For a position loop, *command* and *feedback* are in position units. For a linear axis, this could be inches, mm, meters, or whatever is relevant. Likewise, for an angular axis, it could be degrees, radians, etc. The units of the *output* pin represent the change needed to make the feedback match the command. As such, for a position loop *Output* is a velocity, in inches/sec, mm/sec, degrees/sec, etc. Time units are always seconds, and the velocity units match the position units. If command and feedback are in meters, then output is in meters per second.

Each loop has two pins which are used to monitor or control the general operation of the component.

- (float) *pid.<loopnum>.error* - Equals *.command* minus *.feedback*.
- (bit) *pid.<loopnum>.enable* - A bit that enables the loop. If *.enable* is false, all integrators are reset, and the output is forced to zero. If *.enable* is true, the loop operates normally.

Pins used to report saturation. Saturation occurs when the output of the PID block is at its maximum or minimum limit.

- (bit) *pid.<loopnum>.saturated* - True when output is saturated.
- (float) *pid.<loopnum>.saturated_s* - The time the output has been saturated.
- (s32) *pid.<loopnum>.saturated_count* - The time the output has been saturated.

The PID gains, limits, and other *tunable* features of the loop are available as pins so that they can be adjusted dynamically for more advanced tuning possibilities.

- (float) *pid.<loopnum>.Pgain* - Proportional gain
- (float) *pid.<loopnum>.Igain* - Integral gain
- (float) *pid.<loopnum>.Dgain* - Derivative gain
- (float) *pid.<loopnum>.bias* - Constant offset on output
- (float) *pid.<loopnum>.FF0* - Zeroth order feedforward - output proportional to command (position).
- (float) *pid.<loopnum>.FF1* - First order feedforward - output proportional to derivative of command (velocity).
- (float) *pid.<loopnum>.FF2* - Second order feedforward - output proportional to 2nd derivative of command (acceleration).
- (float) *pid.<loopnum>.deadband* - Amount of error that will be ignored
- (float) *pid.<loopnum>.maxerror* - Limit on error
- (float) *pid.<loopnum>.maxerrorI* - Limit on error integrator
- (float) *pid.<loopnum>.maxerrorD* - Limit on error derivative
- (float) *pid.<loopnum>.maxcmdD* - Limit on command derivative
- (float) *pid.<loopnum>.maxcmdDD* - Limit on command 2nd derivative

- (float) *pid.<loopnum>.maxoutput* - Limit on output value

All *max** limits are implemented so that if the value of this parameter is zero, there is no limit.

If *debug=1* was specified when the component was installed, four additional pins will be exported:

- (float) *pid.<loopnum>.errorI* - Integral of error.
- (float) *pid.<loopnum>.errorD* - Derivative of error.
- (float) *pid.<loopnum>.commandD* - Derivative of the command.
- (float) *pid.<loopnum>.commandDD* - 2nd derivative of the command.

15.11.4.2 Functions

The component exports one function for each PID loop. This function performs all the calculations needed for the loop. Since each loop has its own function, individual loops can be included in different threads and execute at different rates.

- (funct) *pid.<loopnum>.do_pid_calcs* - Performs all calculations for a single PID loop.

If you want to understand the exact algorithm used to compute the output of the PID loop, refer to figure [PID Loop Block Diagram](#), the comments at the beginning of *emc2/src/hal/components/pid.c*, and of course to the code itself. The loop calculations are in the C function *calc_pid()*.

15.11.5 Simulated Encoder

The simulated encoder is exactly that. It produces quadrature pulses with an index pulse, at a speed controlled by a HAL pin. Mostly useful for testing.

Loading sim-encoder

```
halcmd: loadrt sim-encoder num_chan=<number>
```

<number> is the number of encoders that you want to simulate. If not specified, one encoder will be installed. The maximum number is 8 (as defined by *MAX_CHAN* in *sim_encoder.c*).

Unloading sim-encoder

```
halcmd: unloadrt sim-encoder
```

15.11.5.1 Pins

- (float) *sim-encoder.<chan-num>.speed* - The speed command for the simulated shaft.
- (bit) *sim-encoder.<chan-num>.phase-A* - Quadrature output.
- (bit) *sim-encoder.<chan-num>.phase-B* - Quadrature output.
- (bit) *sim-encoder.<chan-num>.phase-Z* - Index pulse output.

When *.speed* is positive, *.phase-A* leads *.phase-B*.

15.11.5.2 Parameters

- (*u32*) *sim-encoder.<chan-num>.ppr* - Pulses Per Revolution.
- (*float*) *sim-encoder.<chan-num>.scale* - Scale Factor for *speed*. The default is 1.0, which means that *speed* is in revolutions per second. Change to 60 for RPM, to 360 for degrees per second, 6.283185 for radians per second, etc.

Note that pulses per revolution is not the same as counts per revolution. A pulse is a complete quadrature cycle. Most encoder counters will count four times during one complete cycle.

15.11.5.3 Functions

The component exports two functions. Each function affects all simulated encoders.

- (*funct*) *sim-encoder.make-pulses* - High speed function to generate quadrature pulses (no floating point).
- (*funct*) *sim-encoder.update-speed* - Low speed function to read *speed*, do scaling, and set up *make-pulses*.

15.11.6 Debounce

Debounce is a realtime component that can filter the glitches created by mechanical switch contacts. It may also be useful in other applications where short pulses are to be rejected.

Loading debounce

```
halcmd: loadrt debounce cfg=<config-string>
```

<config-string>

Is a series of comma separated decimal integers. Each number installs a group of identical debounce filters, the number determines how many filters are in the group.

Loading debounce Example

```
halcmd: loadrt debounce cfg=1,4,2
```

will install three groups of filters. Group 0 contains one filter, group 1 contains four, and group 2 contains two filters. The default value for *<config-string>* is "1" which will install a single group containing a single filter. The maximum number of groups 8 (as defined by MAX_GROUPS in debounce.c). The maximum number of filters in a group is limited only by shared memory space. Each group is completely independent. All filters in a single group are identical, and they are all updated by the same function at the same time. In the following descriptions, *<G>* is the group number and *<F>* is the filter number within the group. The first filter is group 0, filter 0.

Unloading debounce

```
halcmd: unloadrt debounce
```

15.11.6.1 Pins

Each individual filter has two pins.

- (*bit*) *debounce.<G>.<F>.in* - Input of filter *<F>* in group *<G>*.
- (*bit*) *debounce.<G>.<F>.out* - Output of filter *<F>* in group *<G>*.

15.11.6.2 Parameters

Each group of filters has one parameter¹¹.

- *(s32) debounce.<G>.delay* - Filter delay for all filters in group <G>.

The filter delay is in units of thread periods. The minimum delay is zero. The output of a zero delay filter exactly follows its input - it doesn't filter anything. As *delay* increases, longer and longer glitches are rejected. If *delay* is 4, all glitches less than or equal to four thread periods will be rejected.

15.11.6.3 Functions

Each group of filters has one function, which updates all the filters in that group *simultaneously*. Different groups of filters can be updated from different threads at different periods.

- *(funct) debounce.<G>* - Updates all filters in group <G>.

15.11.7 Siggen

Siggen is a realtime component that generates square, triangle, and sine waves. It is primarily used for testing.

Loading siggen

```
halcmd: loadrt siggen [num_chan=<chans>]
```

<chans>

is the number of signal generators that you want to install. If *numchan* is not specified, one signal generator will be installed. The maximum number of generators is 16 (as defined by MAX_CHAN in siggen.c). Each generator is completely independent. In the following descriptions,

<chan>

is the number of a specific signal generator (the numbers start at 0).

Unloading siggen

```
halcmd: unloadrt siggen
```

15.11.7.1 Pins

Each generator has five output pins.

- *(float) siggen.<chan>.sine* - Sine wave output.
- *(float) siggen.<chan>.cosine* - Cosine output.
- *(float) siggen.<chan>.sawtooth* - Sawtooth output.
- *(float) siggen.<chan>.triangle* - Triangle wave output.
- *(float) siggen.<chan>.square* - Square wave output.

¹¹Each individual filter also has an internal state variable. There is a compile time switch that can export that variable as a parameter. This is intended for testing, and simply wastes shared memory under normal circumstances.

All five outputs have the same frequency, amplitude, and offset.

In addition to the output pins, there are three control pins:

- *(float) siggen.<chan>.frequency* - Sets the frequency in Hertz, default value is 1 Hz.
- *(float) siggen.<chan>.amplitude* - Sets the peak amplitude of the output waveforms, default is 1.
- *(float) siggen.<chan>.offset* - Sets DC offset of the output waveforms, default is 0.

For example, if *siggen.0.amplitude* is 1.0 and *siggen.0.offset* is 0.0, the outputs will swing from -1.0 to +1.0. If *siggen.0.amplitude* is 2.5 and *siggen.0.offset* is 10.0, then the outputs will swing from 7.5 to 12.5.

15.11.7.2 Parameters

None. ¹²

15.11.7.3 Functions

- *(funct) siggen.<chan>.update* - Calculates new values for all five outputs.

15.11.8 lut5

The lut5 component is a 5 input logic component based on a look up table.

- *lut5* does not require a floating point thread.

Loading lut5

```
loadrt lut5 [count=N|names=name1[,name2...]]
addf lut5.N servo-thread | base-thread
setp lut5.N.function 0xN
```

Lut5 Computing Function To compute the hexadecimal number for the function starting from the top put a 1 or 0 to indicate if that row would be true or false. Next write down every number in the output column starting from the top and writing them from right to left. This will be the binary number. Using a calculator with a program view like the one in Ubuntu enter the binary number and then convert it to hexadecimal and that will be the value for function.

Table 15.5: Lut5 Look Up Table

Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Output
0	0	0	0	0	
0	0	0	0	1	
0	0	0	1	0	
0	0	0	1	1	
0	0	1	0	0	
0	0	1	0	1	
0	0	1	1	0	
0	0	1	1	1	

¹²Prior to version 2.1, frequency, amplitude, and offset were parameters. They were changed to pins to allow control by other components.

Table 15.5: (continued)

Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Output
0	1	0	0	0	
0	1	0	0	1	
0	1	0	1	0	
0	1	0	1	1	
0	1	1	0	0	
0	1	1	0	1	
0	1	1	1	0	
0	1	1	1	1	
1	0	0	0	0	
1	0	0	0	1	
1	0	0	1	0	
1	0	0	1	1	
1	0	1	0	0	
1	0	1	0	1	
1	0	1	1	0	
1	0	1	1	1	
1	1	0	0	0	
1	1	0	0	1	
1	1	0	1	0	
1	1	0	1	1	
1	1	1	0	0	
1	1	1	0	1	
1	1	1	1	0	
1	1	1	1	1	

Lut5 Two Inputs Example In the following table we have selected the output state for each line that we wish to be true.

Table 15.6: Lut5 Two Inputs Example Look Up Table

Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Output
0	0	0	0	0	0
0	0	0	0	1	1
0	0	0	1	0	0
0	0	0	1	1	1

Looking at the output column of our example we want the output to be on when Bit 0 or Bit 0 and Bit1 is on and nothing else. The binary number is *b1010* (rotate the output 90 degrees CW). Enter this number into the calculator then change the display to hexadecimal and the number needed for function is *0xa*. The hexadecimal prefix is *0x*.

15.12 HAL Examples

All of these examples assume you are starting with a stepconf based configuration and have two threads *base-thread* and *servo-thread*. The stepconf wizard will create an empty *custom.hal* and a *custom_postgui.hal* file. The *custom.hal* file will be loaded after the configuration HAL file and the *custom_postgui.hal* file is loaded after the GUI has been loaded.

15.12.1 Connecting Two Outputs

To connect two outputs to an input you can use the `or2` component. The `or2` works like this, if either input to `or2` is on then the `or2` output is on. If neither input to `or2` is on the `or2` output is off.

For example to have two `pyvcp` buttons both connected to one led.

The .xml file

```
<pyvcp>
  <button>
    <halpin>"button-1"</halpin>
    <text>"Button 1"</text>
  </button>

  <button>
    <halpin>"button-2"</halpin>
    <text>"Button 2"</text>
  </button>

  <led>
    <halpin>"led-1"</halpin>
    <size>50</size>
    <on_color>"green"</on_color>
    <off_color>"red"</off_color>
  </led>
</pyvcp>
```

The postgui.hal file

```
loadrt or2
addf or2.0 servo-thread
net button-1 or2.0.in0 <= pyvcp.button-1
net button-2 or2.0.in1 <= pyvcp.button-2
net led-1 pyvcp.led-1 <= or2.0.out
```

When you run this example in an Axis simulator created with the Stepconf Wizard you can open a terminal and see the pins created with `loadrt or2` by typing in `halcmd show pin or2` in the terminal

```
halcmd show pin or2
Component Pins:
Owner  Type  Dir      Value  Name
   22  bit   IN       FALSE  or2.0.in0 <== button-1
   22  bit   IN       FALSE  or2.0.in1 <== button-2
   22  bit   OUT      FALSE  or2.0.out ==> led-1
```

You can see from the HAL command `show pin or2` that the `button-1` pin is connected to the `or2.0.in0` pin and from the direction arrow you can see that the button is an output and the `or2.0.in0` is an input. The output from `or2` goes to the input of the led.

15.12.2 Manual Toolchange

In this example it is assumed that you're *rolling your own* configuration and wish to add the HAL Manual Toolchange window. The HAL Manual Toolchange is primarily useful if you have presettable tools and you store the offsets in the tool table. If you need to touch off for each tool change then it is best just to split up your G-code. To use the HAL Manual Toolchange window you basically have to load the `hal_manualtoolchange` component then send the `iocontrol tool change` to the `hal_manualtoolchange change` and send the `hal_manualtoolchange changed` back to the `iocontrol tool changed`. A pin is provided for an external input to indicate the tool change is complete.

This is an example of manual toolchange *with* the HAL Manual Toolchange component:

```
loadusr -W hal_manualtoolchange
net tool-change iocontrol.0.tool-change => hal_manualtoolchange.change
net tool-changed iocontrol.0.tool-changed <= hal_manualtoolchange.changed
net external-tool-changed hal_manualtoolchange.change_button <= parport.0.pin-12-in
net tool-number iocontrol.0.tool-prep-number => hal_manualtoolchange.number
net tool-prepare-loopback iocontrol.0.tool-prepare => iocontrol.0.tool-prepared
```

This is an example of manual toolchange *without* the HAL Manual Toolchange component:

```
net tool-number <= iocontrol.0.tool-prep-number
net tool-change-loopback iocontrol.0.tool-change => iocontrol.0.tool-changed
net tool-prepare-loopback iocontrol.0.tool-prepare => iocontrol.0.tool-prepared
```

15.12.3 Compute Velocity

This example uses *ddt*, *mult2* and *abs* to compute the velocity of a single axis. For more information on the real time components see the man pages or the Realtime Components section (Section [15.10.2](#)).

The first thing is to check your configuration to make sure you are not using any of the real time components all ready. You can do this by opening up the HAL Configuration window and look for the components in the pin section. If you are then find the .hal file that they are being loaded in and increase the counts and adjust the instance to the correct value. Add the following to your custom.hal file.

Load the realtime components.

```
loadrt ddt count=1
loadrt mult2 count=1
loadrt abs count=1
```

Add the functions to a thread so it will get updated.

```
addf ddt.0 servo-thread
addf mult2.0 servo-thread
addf abs.0 servo-thread
```

Make the connections.

```
setp mult2.in1 60
net xpos-cmd ddt.0.in
net X-IPS mult2.0.in0 <= ddt.0.out
net X-ABS abs.0.in <= mult2.0.out
net X-IPM abs.0.out
```

In this last section we are setting the mult2.0.in1 to 60 to convert the inch per second to inch per minute that we get from the ddt.0.out.

The xpos-cmd sends the commanded position to the ddt.0.in. The ddt computes the derivative of the change of the input.

The ddt2.0.out is multiplied by 60 to give IPM.

The mult2.0.out is sent to the abs to get the absolute value.

The following figure shows the result when the X axis is moving at 15 IPM in the minus direction. Notice that we can get the absolute value from either the abs.0.out pin or the X-IPM signal.

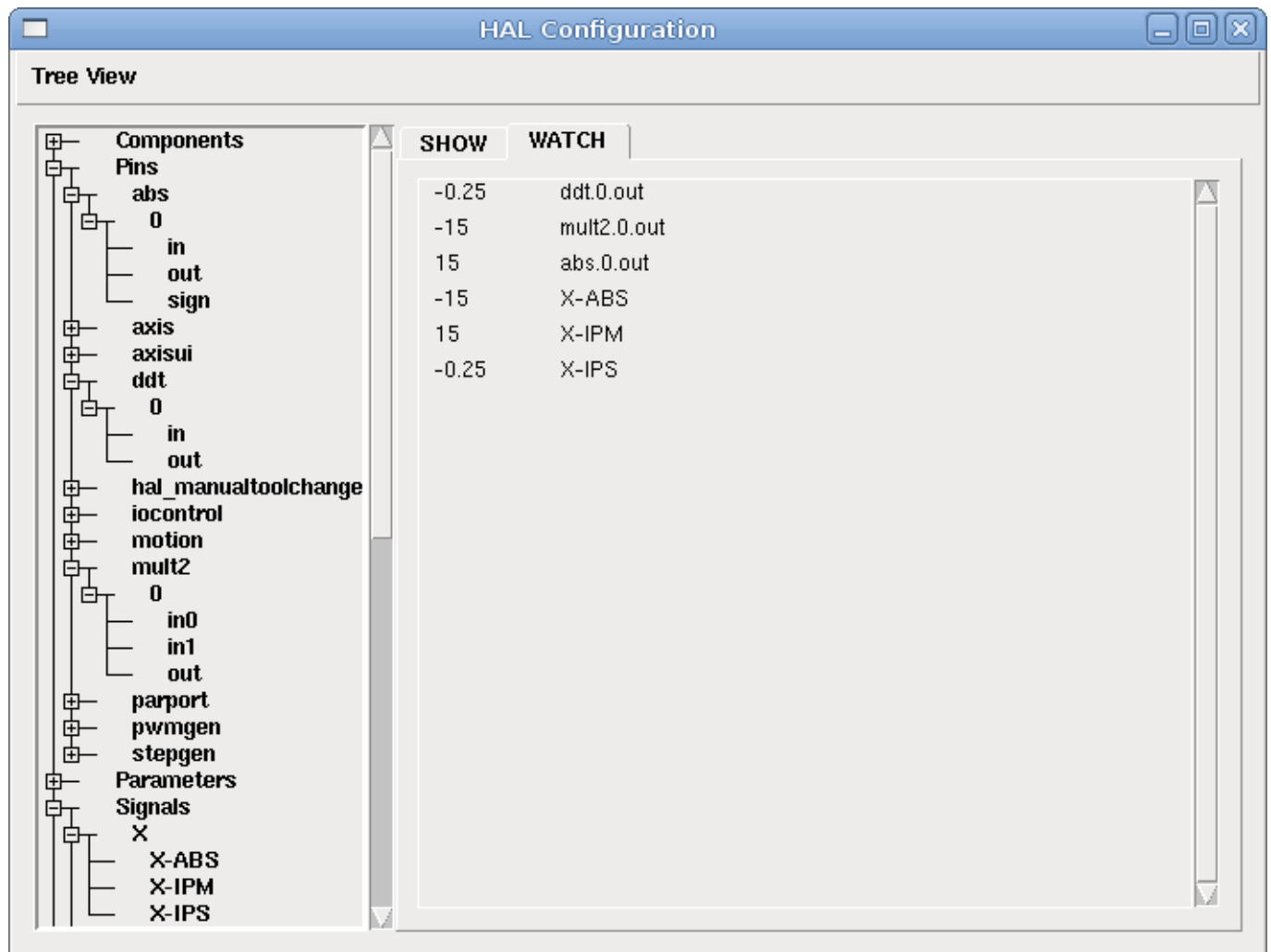


Figure 15.36: HAL: Velocity Example

15.12.4 Soft Start Details

This example shows how the HAL components *lowpass*, *limit2* or *limit3* can be used to limit how fast a signal changes.

In this example we have a servo motor driving a lathe spindle. If we just used the commanded spindle speeds on the servo it will try to go from present speed to commanded speed as fast as it can. This could cause a problem or damage the drive. To slow the rate of change we can send the spindle.N.speed-out through a limiter before the PID, so that the PID command value changes to new settings more slowly.

Three built-in components that limit a signal are:

- *limit2* limits the range and first derivative of a signal.
- *limit3* limits the range, first and second derivatives of a signal.
- *lowpass* uses an exponentially-weighted moving average to track an input signal.

To find more information on these HAL components check the man pages.

Place the following in a text file called `softstart.hal`. If you're not familiar with Linux place the file in your home directory.


```
loadrt threads period1=1000000 name1=thread
loadrt siggen
loadrt lowpass
loadrt limit2
loadrt limit3
net square siggen.0.square => lowpass.0.in limit2.0.in limit3.0.in
net lowpass <= lowpass.0.out
net limit2 <= limit2.0.out
net limit3 <= limit3.0.out
setp siggen.0.frequency .1
setp lowpass.0.gain .01
setp limit2.0.maxv 2
setp limit3.0.maxv 2
setp limit3.0.maxa 10
addf siggen.0.update thread
addf lowpass.0 thread
addf limit2.0 thread
addf limit3.0 thread
start
loadusr halscope
```

Open a terminal window and run the file with the following command.

```
halrun -I softstart.hal
```

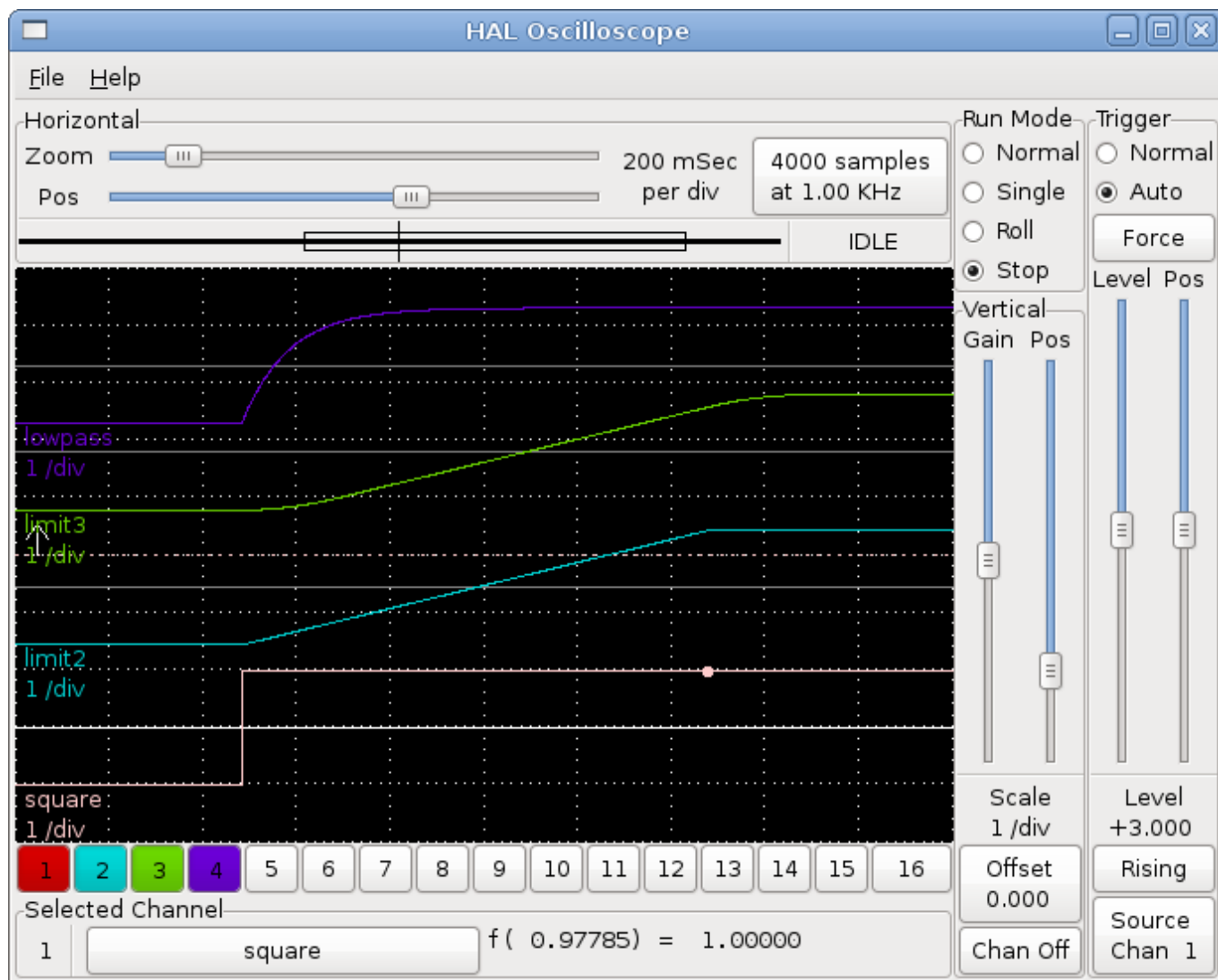
When the HAL Oscilloscope first starts up click *OK* to accept the default thread.

Next you have to add the signals to the channels. Click on channel 1 then select *square* from the Signals tab. Repeat for channels 2-4 and add lowpass, limit2, and limit3.

Next to set up a trigger signal click on the Source None button and select square. The button will change to Source Chan 1.

Next click on Single in the Run Mode radio buttons box. This will start a run and when it finishes you will see your traces.

To separate the signals so you can see them better click on a channel then use the Pos slider in the Vertical box to set the positions.



To see the effect of changing the set point values of any of the components you can change them in the terminal window. To see what different gain settings do for lowpass just type the following in the terminal window and try different settings.

```
setp lowpass.0.gain *.01
```

After changing a setting run the oscilloscope again to see the change.

When you're finished type *exit* in the terminal window to shut down halrun and close the halscope. Don't close the terminal window with halrun running as it might leave some things in memory that could prevent LinuxCNC from loading.

For more information on Halscope see the HAL manual and the tutorial.

15.12.5 Stand Alone HAL

In some cases you might want to run a GladeVCP screen with just HAL. For example say you had a stepper driven device that all you need is to run a stepper motor. A simple *Start/Stop* interface is all you need for your application so no need to load up and configure a full blown CNC application.

In the following example we have created a simple GladeVCP panel with one stepper.

Basic Syntax

```

# load the winder.glade GUI and name it winder
loadusr -Wn winder gladevcp -c winder -u handler.py winder.glade

# load realtime components
loadrt threads name1=fast period1=50000 fpl=0 name2=slow period2=1000000
loadrt stepgen step_type=0 ctrl_type=v
loadrt hal_parport cfg="0x378 out"

# add functions to threads
addf stepgen.make-pulses fast
addf stepgen.update-freq slow
addf stepgen.capture-position slow
addf parport.0.read fast
addf parport.0.write fast

# make HAL connections
net winder-step parport.0.pin-02-out <= stepgen.0.step
net winder-dir parport.0.pin-03-out <= stepgen.0.dir
net run-stepgen stepgen.0.enable <= winder.start_button

# start the threads
start

# comment out the following lines while testing and use the interactive
# option halrun -I -f start.hal to be able to show pins etc.

# wait until the GladeVCP GUI named winder terminates
waitusr winder

# stop HAL threads
stop

# unload HAL all components before exiting
unloadrt all

```

15.13 HAL Component Generator

15.13.1 Introduction

Writing a HAL component can be a tedious process, most of it in setup calls to *rtapi_* and *hal_* functions and associated error checking. *halcompile* will write all this code for you, automatically.

Compiling a HAL component is also much easier when using *halcompile*, whether the component is part of the LinuxCNC source tree, or outside it.

For instance, when coded in C, a simple component such as "ddt" is around 80 lines of code. The equivalent component is very short when written using the *halcompile* preprocessor:

Simple Component Example

```

component ddt "Compute the derivative of the input function";
pin in float in;
pin out float out;
variable double old;
function _;
license "GPL"; // indicates GPL v2 or later
;;
float tmp = in;
out = (tmp - old) / fperiod;

```

```
old = tmp;
```

15.13.2 Installing

To compile a component, if a packaged version of LinuxCNC is used, development packages have to be installed using either Synaptic from the main menu *System -> Administration -> Synaptic package manager* or running one of the following commands in a terminal window:

Development packages installation

```
sudo apt install linuxcnc-dev  
# or  
sudo apt install linuxcnc-ospace-dev
```

Another method is using the Synaptic package manager, from the Applications menu, to install the `linuxcnc-dev` or `linuxcnc-ospace-dev` packages.

15.13.3 Using a Component

Components need to be loaded and added to a thread before it can be used.

Example

```
loadrt threads name1=servo-thread period1=1000000  
loadrt ddt  
addf ddt.0 servo-thread
```

More information on *loadrt* and *addf* can be found in the [HAL Basics](#).

To test your component you can follow the examples in the [HAL Tutorial](#).

15.13.4 Definitions

- *component* - A component is a single real-time module, which is loaded with *halcmd loadrt*. One *.comp* file specifies one component. The component name and file name must match.
- *instance* - A component can have zero or more instances. Each instance of a component is created equal (they all have the same pins, parameters, functions, and data) but behave independently when their pins, parameters, and data have different values.
- *singleton* - It is possible for a component to be a "singleton", in which case exactly one instance is created. It seldom makes sense to write a *singleton* component, unless there can literally only be a single object of that kind in the system (for instance, a component whose purpose is to provide a pin with the current UNIX time, or a hardware driver for the internal PC speaker)

15.13.5 Instance creation

For a singleton, the one instance is created when the component is loaded.

For a non-singleton, the *count* module parameter determines how many numbered instances are created. If *count* is not specified, the *names* module parameter determines how many named instances are created. If neither *count* nor *names* is specified, a single numbered instance is created.

15.13.6 Implicit Parameters

Functions are implicitly passed the *period* parameter which is the time in nanoseconds of the last period to execute the component. Functions which use floating-point can also refer to *fperiod* which is the floating-point time in seconds, or (period*1e-9). This can be useful in components that need the timing information.

15.13.7 Syntax

A *.comp* file consists of a number of declarations, followed by `;;` on a line of its own, followed by C code implementing the module's functions.

Declarations include:

- *component* HALNAME (DOC);
- *pin* PINDIRECTION TYPE HALNAME ([SIZE][MAXSIZE: CONDSIZE]) (if CONDITION) (= START-VALUE) (DOC) ;
- *param* PARAMDIRECTION TYPE HALNAME ([SIZE][MAXSIZE: CONDSIZE]) (if CONDITION) (= STARTVALUE) (DOC) ;
- *function* HALNAME (fp | nofp) (DOC);
- *option* OPT (VALUE);
- *variable* CTYPE STARREDNAME ([SIZE]);
- *description* DOC;
- *examples* DOC;
- *notes* DOC;
- *see_also* DOC;
- *license* LICENSE;
- *author* AUTHOR;
- *include* HEADERFILE;

Parentheses indicate optional items. A vertical bar indicates alternatives. Words in *CAPITALS* indicate variable text, as follows:

- *NAME* - A standard C identifier
- *STARREDNAME* - A C identifier with zero or more * before it. This syntax can be used to declare instance variables that are pointers. Note that because of the grammar, there may not be whitespace between the * and the variable name.
- *HALNAME* - An extended identifier. When used to create a HAL identifier, any underscores are replaced with dashes, and any trailing dash or period is removed, so that "this_name_" will be turned into "this-name", and if the name is "_", then a trailing period is removed as well, so that "function_" gives a HAL function name like "component.<num>" instead of "component.<num>." If present, the prefix *hal_* is removed from the beginning of the component name when creating pins, parameters and functions.

In the HAL identifier for a pin or parameter, # denotes an array item, and must be used in conjunction with a *[SIZE]* declaration. The hash marks are replaced with a 0-padded number with the same length as the number of # characters.

When used to create a C identifier, the following changes are applied to the HALNAME:

1. Any "#" characters, and any ".", "_" or "-" characters immediately before them, are removed.
2. Any remaining "." and "-" characters are replaced with "_".
3. Repeated "_" characters are changed to a single "_" character.

A trailing "_" is retained, so that HAL identifiers which would otherwise collide with reserved names or keywords (e.g., *min*) can be used.

HALNAME	C Identifier	HAL Identifier
x_y_z	x_y_z	x-y-z
x-y.z	x_y_z	x-y.z
x_y_z_	x_y_z_	x-y-z
x.##.y	x_y(MM)	x.MM.z
x.##	x(MM)	x.MM

- *if CONDITION* - An expression involving the variable *personality* which is nonzero when the pin or parameter should be created
- *SIZE* - A number that gives the size of an array. The array items are numbered from 0 to *SIZE*-1.
- *MAXSIZE : CONDSIZE* - A number that gives the maximum size of the array followed by an expression involving the variable *personality* and which always evaluates to less than *MAXSIZE*. When the array is created its size will be *CONDSIZE*.
- *DOC* - A string that documents the item. String can be a C-style "double quoted" string, like:

```
"Selects the desired edge: TRUE means falling, FALSE means rising"
```

or a Python-style "triple quoted" string, which may include embedded newlines and quote characters, such as:

```
"""The effect of this parameter, also known as "the orb of zot",
will require at least two paragraphs to explain.

Hopefully these paragraphs have allowed you to understand "zot"
better."""
```

Or a string may be preceded by the literal character *r*, in which case the string is interpreted like a Python raw-string.

The documentation string is in "groff-man" format. For more information on this markup format, see *groff_man(7)*. Remember that *halcompile* interprets backslash escapes in strings, so for instance to set the italic font for the word *example*, write:

```
"\\fIexample\\fB"
```

In this case, r-strings are particularly useful, because the backslashes in an r-string need not be doubled:

```
r"\\fIexample\\fB"
```

- *TYPE* - One of the HAL types: *bit*, *signed*, *unsigned*, or *float*. The old names *s32* and *u32* may also be used, but *signed* and *unsigned* are preferred.

- *PINDIRECTION* - One of the following: *in*, *out*, or *io*. A component sets a value for an *out* pin, it reads a value from an *in* pin, and it may read or set the value of an *io* pin.
- *PARAMDIRECTION* - One of the following: *r* or *rw*. A component sets a value for a *r* parameter, and it may read or set the value of a *rw* parameter.
- *STARTVALUE* - Specifies the initial value of a pin or parameter. If it is not specified, then the default is 0 or *FALSE*, depending on the type of the item.
- *HEADERFILE* - The name of a header file, either in double-quotes (include "myfile.h";) or in angle brackets (include <systemfile.h>;). The header file will be included (using C's #include) at the top of the file, before pin and parameter declarations.

15.13.7.1 HAL functions

- *fp* - Indicates that the function performs floating-point calculations.
- *nofp* - Indicates that it only performs integer calculations. If neither is specified, *fp* is assumed. Neither *halcompile* nor gcc can detect the use of floating-point calculations in functions that are tagged *nofp*, but use of such operations results in undefined behavior.

15.13.7.2 Options

The currently defined options are:

- *option singleton yes* - (default: no) Do not create a *count* module parameter, and always create a single instance. With *singleton*, items are named *component-name.item-name* and without *singleton*, items for numbered instances are named *component-name.<num>.item-name*.
- *option default_count number* - (default: 1) Normally, the module parameter *count* defaults to 1. If specified, the *count* will default to this value instead.
- *option count_function yes* - (default: no) Normally, the number of instances to create is specified in the module parameter *count*; if *count_function* is specified, the value returned by the function *int get_count(void)* is used instead, and the *count* module parameter is not defined.
- *option rtapi_app no* - (default: yes) Normally, the functions *rtapi_app_main()* and *rtapi_app_exit()* are automatically defined. With *option rtapi_app no*, they are not, and must be provided in the C code. Use the following prototypes:

```
'int rtapi_app_main(void);'
'void rtapi_app_exit(void);'
```

When implementing your own *rtapi_app_main()*, call the function *int export(char *prefix, long extra_arg)* to register the pins, parameters, and functions for *prefix*.

- *option data TYPE* - (default: none) **deprecated** If specified, each instance of the component will have an associated data block of type *TYPE* (which can be a simple type like *float* or the name of a type created with *typedef*). In new components, *variable* should be used instead.
- *option extra_setup yes* - (default: no) If specified, call the function defined by *EXTRA_SETUP* for each instance. If using the automatically defined *rtapi_app_main*, *extra_arg* is the number of this instance.
- *option extra_cleanup yes* - (default: no) If specified, call the function defined by *EXTRA_CLEANUP* from the automatically defined *rtapi_app_exit*, or if an error is detected in the automatically defined *rtapi_app_main*.

- *option userspace yes* - (default: no) If specified, this file describes a userspace (i.e., non-realtime) component, rather than a regular (i.e., realtime) one. A userspace component may not have functions defined by the *function* directive. Instead, after all the instances are constructed, the C function `void user_mainloop(void);` is called. When this function returns, the component exits. Typically, *user_mainloop()* will use *FOR_ALL_INSTS()* to perform the update action for each instance, then sleep for a short time. Another common action in *user_mainloop()* may be to call the event handler loop of a GUI toolkit.
- *option userinit yes* - (default: no) This option is ignored if the option *userspace* (see above) is set to *no*. If *userinit* is specified, the function *userinit(argc,argv)* is called before *rtapi_app_main()* (and thus before the call to *hal_init()*). This function may process the commandline arguments or take other actions. Its return type is *void*; it may call *exit()* if it wishes to terminate rather than create a HAL component (for instance, because the commandline arguments were invalid).
- *option extra_link_args "..."* - (default: "") This option is ignored if the option *userspace* (see above) is set to *no*. When linking a userspace component, the arguments given are inserted in the link line. Note that because compilation takes place in a temporary directory, "-L." refers to the temporary directory and not the directory where the .comp source file resides.
- *option extra_compile_args "..."* - (default: "") This option is ignored if the option *userspace* (see above) is set to *no*. When compiling a userspace component, the arguments given are inserted in the compiler command line.
- *option homemod yes* - (default: no) Module is a custom Homing module loaded using [EMCMOT]HOMEMOD=modulename
- *option tpmmod yes* - (default: no) Module is a custom Trajectory Planning (tp) module loaded using [TRAJ]TPMOD=modulename

If an option's VALUE is not specified, then it is equivalent to specifying *option ... yes*.

The result of assigning an inappropriate value to an option is undefined.

The result of using any other option is undefined.

15.13.7.3 License and Authorship

- *LICENSE* - Specify the license of the module for the documentation and for the *MODULE_LICENSE()* module declaration. For example, to specify that the module's license is GPL v2 or later:

```
license "GPL"; // indicates GPL v2 or later
```

For additional information on the meaning of *MODULE_LICENSE()* and additional license identifiers, see `<linux/module.h>` or the manual page *rtapi_module_param(3)*.

This declaration is **required**.

- *AUTHOR* - Specify the author of the module for the documentation.

15.13.7.4 Per-instance data storage

- `variable CTYPE STARREDNAME; + variable CTYPE STARREDNAME[SIZE]; + variable CTYPE STARREDNAME[SIZE] = DEFAULT; + variable CTYPE STARREDNAME[SIZE] = DEFAULT;`

Declare a per-instance variable *STARREDNAME* of type *CTYPE*, optionally as an array of *SIZE* items, and optionally with a default value *DEFAULT*.

Items with no *DEFAULT* are initialized to all-bits-zero.

CTYPE is a simple one-word C type, such as *float*, *u32*, *s32*, *int*, etc.

Access to array variables uses square brackets.

If a variable is to be of a pointer type, there may not be any space between the `""` and the variable name.

Therefore, the following is acceptable:

```
variable int *example;
```

But the following are not:

```
variable int* badexample;  
variable int * badexample;
```

15.13.7.5 Comments

C++-style one-line comments (`//...`) and C-style multi-line comments (`/* ... */`) are both supported in the declaration section.

15.13.8 Restrictions

Though HAL permits a pin, a parameter, and a function to have the same name, *halcompile* does not. Variable and function names that can not be used or are likely to cause problems include:

- Anything beginning with `_comp`.
- `comp_id`
- `fperiod`
- `rtapi_app_main`
- `rtapi_app_exit`
- `extra_setup`
- `extra_cleanup`

15.13.9 Convenience Macros

Based on the items in the declaration section, *halcompile* creates a C structure called `struct __comp_state`. However, instead of referring to the members of this structure (e.g., `*(inst->name)`), they will generally be referred to using the macros below. The details of `struct __comp_state` and these macros may change from one version of *halcompile* to the next.

- **FUNCTION(name)** - Use this macro to begin the definition of a realtime function which was previously declared with *function NAME*. The function includes a parameter *period* which is the integer number of nanoseconds between calls to the function.
 - **EXTRA_SETUP()** - Use this macro to begin the definition of the function called to perform extra setup of this instance. Return a negative Unix *errno* value to indicate failure (e.g., `return -EBUSY` on failure to reserve an I/O port), or 0 to indicate success.
 - **EXTRA_CLEANUP()** - Use this macro to begin the definition of the function called to perform extra cleanup of the component. Note that this function must clean up all instances of the component, not just one. The `"pin_name"`, `"parameter_name"`, and `"data"` macros may not be used here.
-

- *pin_name* or *parameter_name* - For each pin *pin_name* or param *parameter_name* there is a macro which allows the name to be used on its own to refer to the pin or parameter. When *pin_name* or *parameter_name* is an array, the macro is of the form *pin_name(idx)* or *param_name(idx)* where *idx* is the index into the pin array. When the array is a variable-sized array, it is only legal to refer to items up to its *condsize*.

When the item is a conditional item, it is only legal to refer to it when its *condition* evaluated to a nonzero value.

- *variable_name* - For each variable *variable_name* there is a macro which allows the name to be used on its own to refer to the variable. When *variable_name* is an array, the normal C-style subscript is used: *variable_name[idx]*.
- *data* - If "option data" is specified, this macro allows access to the instance data.
- *fperiod* - The floating-point number of seconds between calls to this realtime function.
- *FOR_ALL_INSTS() {...}* - For userspace components. This macro iterates over all the defined instances. Inside the body of the loop, the *pin_name*, *parameter_name*, and *data* macros work as they do in realtime functions.

15.13.10 Components with one function

If a component has only one function and the string "FUNCTION" does not appear anywhere after `;;`, then the portion after `;;` is all taken to be the body of the component's single function. See the [Simple Comp](#) for an example of this.

15.13.11 Component Personality

If a component has any pins or parameters with an "if condition" or "[maxsize : condsizel]", it is called a component with *personality*. The *personality* of each instance is specified when the module is loaded. *Personality* can be used to create pins only when needed. For instance, *personality* is used in the *logic* component, to allow for a variable number of input pins to each logic gate and to allow for a selection of any of the basic boolean logic functions *and*, *or*, and *xor*.

The default number of allowed *personality* items is a compile-time setting (64). The default applies to numerous components included in the distribution that are built using *halcompile*.

To alter the allowed number of personality items for user-built components, use the `--personality` option with *halcompile*. For example, to allow up to 128 personality times:

```
[sudo] halcompile --personality=128 --install ...
```

When using components with personality, normal usage is to specify a personality item for **each** specified component instance. Example for 3 instances of the logic component:

```
loadrt logic names=and4,or3,nand5, personality=0x104,0x203,0x805
```

Note

If a *loadrt* line specifies more instances than personalities, the instances with unspecified personalities are assigned a personality of 0. If the requested number of instances exceeds the number of allowed personalities, personalities are assigned by indexing modulo the number of allowed personalities. A message is printed denoting such assignments.

15.13.12 Compiling

Place the *.comp* file in the source directory *linuxcnc/src/hal/components* and re-run *make*. *Comp* files are automatically detected by the build system.

If a *.comp* file is a driver for hardware, it may be placed in *linuxcnc/src/hal/drivers* and will be built unless LinuxCNC is configured as a userspace simulator.

15.13.13 Compiling realtime components outside the source tree

halcompile can process, compile, and install a realtime component in a single step, placing *rtexample.ko* in the LinuxCNC realtime module directory:

```
[sudo] halcompile --install rtexample.comp
```

Note

sudo (for root permission) is needed when using LinuxCNC from a deb package install. When using a Run-In-Place (RIP) build, root privileges should not be needed.

Or, it can process and compile in one step, leaving *example.ko* (or *example.so* for the simulator) in the current directory:

```
halcompile --compile rtexample.comp
```

Or it can simply process, leaving *example.c* in the current directory:

```
halcompile rtexample.comp
```

halcompile can also compile and install a component written in C, using the *--install* and *--compile* options shown above:

```
[sudo] halcompile --install rtexample2.c
```

man-format documentation can also be created from the information in the declaration section:

```
halcompile --document rtexample.comp
```

The resulting manpage, *example.9* can be viewed with

```
man ./example.9
```

or copied to a standard location for manual pages.

15.13.14 Compiling userspace components outside the source tree

halcompile can process, compile, install, and document userspace components:

```
halcompile usrexample.comp
halcompile --compile usrexample.comp
[sudo] halcompile --install usrexample.comp
halcompile --document usrexample.comp
```

This only works for *.comp* files, not for *.c* files.

15.13.15 Examples

15.13.15.1 constant

Note that the declaration "function _" creates functions named "constant.0", etc. The file name must match the component name.

```
component constant;
pin out float out;
param r float value = 1.0;
function _;
license "GPL"; // indicates GPL v2 or later
;;
FUNCTION(_) { out = value; }
```

15.13.15.2 sincos

This component computes the sine and cosine of an input angle in radians. It has different capabilities than the "sine" and "cosine" outputs of siggen, because the input is an angle, rather than running freely based on a "frequency" parameter.

The pins are declared with the names *sin_* and *cos_* in the source code so that they do not interfere with the functions *sin()* and *cos()*. The HAL pins are still called *sincos.<num>.sin*.

```
component sincos;
pin out float sin_;
pin out float cos_;
pin in float theta;
function _;
license "GPL"; // indicates GPL v2 or later
;;
#include <rtapi_math.h>
FUNCTION(_) { sin_ = sin(theta); cos_ = cos(theta); }
```

15.13.15.3 out8

This component is a driver for a *fictional* card called "out8", which has 8 pins of digital output which are treated as a single 8-bit value. There can be a varying number of such cards in the system, and they can be at various addresses. The pin is called *out_* because *out* is an identifier used in *<asm/io.h>*. It illustrates the use of *EXTRA_SETUP* and *EXTRA_CLEANUP* to request an I/O region and then free it in case of error or when the module is unloaded.

```
component out8;
pin out unsigned out_ "Output value; only low 8 bits are used";
param r unsigned ioaddr;

function _;

option count_function;
option extra_setup;
option extra_cleanup;
option constructable no;

license "GPL"; // indicates GPL v2 or later
;;
#include <asm/io.h>
```

```

#define MAX 8
int io[MAX] = {0,};
RTAPI_MP_ARRAY_INT(io, MAX, "I/O addresses of out8 boards");

int get_count(void) {
    int i = 0;
    for(i=0; i<MAX && io[i]; i++) { /* Nothing */ }
    return i;
}

EXTRA_SETUP() {
    if(!rtapi_request_region(io[extra_arg], 1, "out8")) {
        // set this I/O port to 0 so that EXTRA_CLEANUP does not release the IO
        // ports that were never requested.
        io[extra_arg] = 0;
        return -EBUSY;
    }
    ioaddr = io[extra_arg];
    return 0;
}

EXTRA_CLEANUP() {
    int i;
    for(i=0; i < MAX && io[i]; i++) {
        rtapi_release_region(io[i], 1);
    }
}

FUNCTION(_) { outb(out_, ioaddr); }

```

15.13.15.4 hal_loop

```

component hal_loop;
pin out float example;

```

This fragment of a component illustrates the use of the *hal_* prefix in a component name. *loop* is the name of a standard Linux kernel module, so a *loop* component might not successfully load if the Linux *loop* module was also present on the system.

When loaded, *halcmd show comp* will show a component called *hal_loop*. However, the pin shown by *halcmd show pin* will be *loop.0.example*, not *hal-loop.0.example*.

15.13.15.5 arraydemo

This realtime component illustrates use of fixed-size arrays:

```

component arraydemo "4-bit Shift register";
pin in bit in;
pin out bit out-# [4];
function _ nofp;
license "GPL"; // indicates GPL v2 or later
;;
int i;
for(i=3; i>0; i--) out(i) = out(i-1);
out(0) = in;

```

15.13.15.6 rand

This userspace component changes the value on its output pin to a new random value in the range (0,1) about once every 1ms.

```
component rand;
option userspace;

pin out float out;
license "GPL"; // indicates GPL v2 or later
;;
#include <unistd.h>

void user_mainloop(void) {
    while(1) {
        usleep(1000);
        FOR_ALL_INSTS() out = drand48();
    }
}
```

15.13.15.7 logic

This realtime component shows how to use "personality" to create variable-size arrays and optional pins.

```
component logic "LinuxCNC HAL component providing experimental logic functions";
pin in bit in-##[16 : personality & 0xff];
pin out bit and if personality & 0x100;
pin out bit or if personality & 0x200;
pin out bit xor if personality & 0x400;
function _ nofp;
description ""
Experimental general 'logic function' component. Can perform 'and', 'or'
and 'xor' of up to 16 inputs. Determine the proper value for 'personality'
by adding:
.IP \\\(bu 4
The number of input pins, usually from 2 to 16
.IP \\\(bu
256 (0x100) if the 'and' output is desired
.IP \\\(bu
512 (0x200) if the 'or' output is desired
.IP \\\(bu
1024 (0x400) if the 'xor' (exclusive or) output is desired"";
license "GPL"; // indicates GPL v2 or later
;;
FUNCTION(_) {
    int i, a=1, o=0, x=0;
    for(i=0; i < (personality & 0xff); i++) {
        if(in(i)) { o = 1; x = !x; }
        else { a = 0; }
    }
    if(personality & 0x100) and = a;
    if(personality & 0x200) or = o;
    if(personality & 0x400) xor = x;
}
```

A typical load line for this component might be

```
loadrt logic count=3 personality=0x102,0x305,0x503
```

which creates the following pins:

- A 2-input AND gate: logic.0.and, logic.0.in-00, logic.0.in-01
- 5-input AND and OR gates: logic.1.and, logic.1.or, logic.1.in-00, logic.1.in-01, logic.1.in-02, logic.1.in-03, logic.1.in-04,
- 3-input AND and XOR gates: logic.2.and, logic.2.xor, logic.2.in-00, logic.2.in-01, logic.2.in-02

15.13.15.8 General Functions

This example shows how to call functions from the main function. It also shows how to pass reference of HAL pins to those functions.

```
component example;
pin in s32 in;
pin out bit out1;
pin out bit out2;

function _;
license "GPL";
;;

// general pin set true function
void set(hal_bit_t *p){
    *p = 1;
}

// general pin set false function
void unset(hal_bit_t *p){
    *p = 0;
}

//main function
FUNCTION(_) {
    if (in < 0){
        set(&out1);
        unset(&out2);
    }else if (in > 0){
        unset(&out2);
        set(&out2);
    }else{
        unset(&out1);
        unset(&out2);
    }
}
```

This component uses two general function to manipulate a HAL bit pin referenced to it.

15.13.16 Command Line Usage

The halcompile man page gives details for invoking halcompile.

```
$ man halcompile
```

A brief summary of halcompile usage is given by:

```
$ halcompile --help
```

15.14 HALTCL Files

The halcmd language excels in specifying components and connections but offers no computational capabilities. As a result, INI files are limited in the clarity and brevity that is possible with higher level languages.

The haltcl facility provides a means to use Tcl scripting and its features for computation, looping, branching, procedures, etc. in INI files. To use this functionality, you use the Tcl language and the extension .tcl for HAL files.

The .tcl extension is understood by the main script (linuxcnc) that processes INI files. Haltcl files are identified in the the HAL section of INI files (just like .hal files).

Example

```
[HAL]
HALFILE = conventional_file.hal
HALFILE = tcl_based_file.tcl
```

With appropriate care, .hal and .tcl files can be intermixed.

15.14.1 Compatibility

The halcmd language used in .hal files has a simple syntax that is actually a subset of the more powerful general-purpose Tcl scripting language.

15.14.2 Haltcl Commands

Haltcl files use the Tcl scripting language augmented with the specific commands of the LinuxCNC hardware abstraction layer (HAL). The HAL-specific commands are:

```
addf, alias,
delf, delsig,
getp, gets
ptype,
stype,
help,
linkpp, linkps, linksp, list, loadrt, loadusr, lock,
net, newsig,
save, setp, sets, show, source, start, status, stop,
unalias, unlinkp, unload, unloadrt, unloadusr, unlock,
waitusr
```

Two special cases occur for the *gets* and *list* commands due to conflicts with Tcl builtin commands. For haltcl, these commands must be preceded with the keyword *hal*:

```
halcmd    haltcl
-----    -----
gets      hal gets
list      hal list
```

15.14.3 Haltcl INI-file variables

INI file variables are accessible by both halcmd and haltcl but with differing syntax.

LinuxCNC INI files use SECTION and ITEM specifiers to identify configuration items:


```
[SECTION_A]
ITEM1 = value_1
ITEM2 = value_2
...
[SECTION_B]
...
```

The INI file values are accessible by text substitution in .hal files using the form:

```
[SECTION]ITEM
```

The same INI file values are accessible in .tcl files using the form of a tcl global array variable:

```
$::SECTION(ITEM)
```

For example, an INI file item like:

```
[JOINT_0]
MAX_VELOCITY = 4
```

is expressed as [JOINT_0]MAX_VELOCITY in .hal files for halcmd and as \$::JOINT_0(MAX_VELOCITY) in .tcl files for haltcl.

Because INI files can repeat the same ITEM in the same SECTION multiple times, \$::SECTION(ITEM) is actually a Tcl list of each individual value.

When there is just one value and it is a simple value (all values that are just letters and numbers without whitespace are in this group), then it is possible to treat \$::SECTION(ITEM) as though it is not a list.

When the value could contain special characters—quote characters, curly-brace characters, embedded whitespace, and other characters that have special meaning in Tcl—it is necessary to distinguish between the list of values and the initial (and possibly only) value in the list.

In Tcl, this is written [lindex \$::SECTION(ITEM) 0].

For example: given the following INI values

```
[HOSTMOT2]
DRIVER=hm2_eth
IPADDR="10.10.10.10"
BOARD=7i92
CONFIG="num_encoders=0 num_pwmgens=0 num_stepgens=6"
```

And this loadrt command:

```
loadrt $::HOSTMOT2(DRIVER) board_ip=$::HOSTMOT2(IPADDR) config=$::HOSTMOT2(CONFIG)
```

Here is the actual command that is run:

```
loadrt hm2_eth board_ip={"10.10.10.10"} config={"num_encoders=0 num_pwmgens=0 num_stepgens ←
=6"}
```

This fails because loadrt does not recognize the braces.

So to get the values just as entered in the INI file, re-write the loadrt line like this:

```
loadrt $::HOSTMOT2(DRIVER) board_ip=[lindex $::HOSTMOT2(IPADDR) 0] config=[lindex ←
$::HOSTMOT2(CONFIG) 0]
```

15.14.4 Converting .hal files to .tcl files

Existing .hal files can be converted to .tcl files by hand editing to adapt to the differences mentioned above. The process can be automated with scripts that convert using these substitutions.

```
[SECTION]ITEM ---> $::SECTION(ITEM)
gets          ---> hal gets
list          ---> hal list
```

15.14.5 Haltcl Notes

In haltcl, the value argument for the *sets* and *setp* commands is implicitly treated as an expression in the Tcl language.

Example

```
# set gain to convert deg/sec to units/min for JOINT_0 radius
setp scale.0.gain 6.28/360.0*$::JOINT_0(radius)*60.0
```

Whitespace in the bare expression is not allowed, use quotes for that:

```
setp scale.0.gain "6.28 / 360.0 * $::JOINT_0(radius) * 60.0"
```

In other contexts, such as *loadrt*, you must explicitly use the Tcl expr command ([*expr* {}]) for computational expressions.

Example

```
loadrt motion base_period=[expr {500000000/$::TRAJ(MAX_PULSE_RATE)}]
```

15.14.6 Haltcl Examples

Consider the topic of *stepgen headroom*. Software stepgen runs best with an acceleration constraint that is "a bit higher" than the one used by the motion planner. So, when using halcmd files, we force INI files to have a manually calculated value.

```
[JOINT_0]
MAXACCEL = 10.0
STEPGEN_MAXACCEL = 10.5
```

With haltcl, you can use Tcl commands to do the computation and eliminate the STEPGEN_MAXACCEL INI file item altogether:

```
setp stepgen.0.maxaccel $::JOINT_0(MAXACCEL)*1.05
```

Another haltcl feature is looping and testing. For example, many simulator configurations use "core_sim.hal" or "core_sim9.hal" hal files. These differ because of the requirement to connect more or fewer axes. The following haltcl code would work for any combination of axes in a trivkins machine.

```
# Create position, velocity and acceleration signals for each axis
set ddt 0
for {set jnum 0} {$jnum < $::KINS(JOINTS)} {incr jnum} {
  # 'list pin' returns an empty list if the pin doesn't exist
  if {[hal list pin joint.${jnum}.motor-pos-cmd] == {}} {
    continue
  }
  net ${jnum}pos joint.${jnum}.motor-pos-cmd => joint.$axno.motor-pos-fb \
    => ddt.$ddt.in
```

```

net ${axis}vel <= ddt.$ddt.out
incr ddt
net ${axis}vel => ddt.$ddt.in
net ${axis}acc <= ddt.$ddt.out
incr ddt
}
puts [show sig *vel]
puts [show sig *acc]

```

15.14.7 Haltcl Interactive

The halrun command recognizes haltcl files. With the -T option, haltcl can be run interactively as a Tcl interpreter. This capability is useful for testing and for standalone hal applications.

Example

```
$ halrun -T haltclfile.tcl
```

15.14.8 Haltcl Distribution Examples (sim)

The configs/sim/axis/simtcl directory includes an INI file that uses a .tcl file to demonstrate a haltcl configuration in conjunction with the usage of twopass processing. The example shows the use of Tcl procedures, looping, the use of comments and output to the terminal.

15.15 Creating Userspace Python Components

15.15.1 Basic usage

A userspace component begins by creating its pins and parameters, then enters a loop which will periodically drive all the outputs from the inputs. The following component copies the value seen on its input pin (*passthrough.in*) to its output pin (*passthrough.out*) approximately once per second.

```

#!/usr/bin/env python3
import hal, time
h = hal.component("passthrough")
h.newpin("in", hal.HAL_FLOAT, hal.HAL_IN)
h.newpin("out", hal.HAL_FLOAT, hal.HAL_OUT)
h.ready()
try:
    while 1:
        time.sleep(1)
        h['out'] = h['in']
except KeyboardInterrupt:
    raise SystemExit

```

Copy the above listing into a file named "passthrough", make it executable (*chmod +x*), and place it on your *\$PATH*. Then try it out:

```

halrun
halcmd: loadusr passthrough
halcmd: show pin

```

```

Component Pins:
Owner Type Dir      Value Name
03  float IN        0  passthrough.in
03  float OUT        0  passthrough.out

```

```
halcmd: setp passthrough.in 3.14
```

```
halcmd: show pin
```

```

Component Pins:
Owner Type Dir      Value Name
03  float IN        3.14 passthrough.in
03  float OUT        3.14 passthrough.out

```

15.15.2 Userspace components and delays

If you typed “show pin” quickly, you may see that *passthrough.out* still had its old value of 0. This is because of the call to *time.sleep(1)*, which makes the assignment to the output pin occur at most once per second. Because this is a userspace component, the actual delay between assignments can be much longer if the memory used by the passthrough component is swapped to disk, the assignment could be delayed until that memory is swapped back in.

Thus, userspace components are suitable for user-interactive elements such as control panels (delays in the range of milliseconds are not noticed, and longer delays are acceptable), but not for sending step pulses to a stepper driver board (delays must always be in the range of microseconds, no matter what).

15.15.3 Create pins and parameters

```
h = hal.component("passthrough")
```

The component itself is created by a call to the constructor *hal.component*. The arguments are the HAL component name and (optionally) the prefix used for pin and parameter names. If the prefix is not specified, the component name is used.

```
h.newpin("in", hal.HAL_FLOAT, hal.HAL_IN)
```

Then pins are created by calls to methods on the component object. The arguments are: pin name suffix, pin type, and pin direction. For parameters, the arguments are: parameter name suffix, parameter type, and parameter direction.

Table 15.7: HAL Option Names

Pin and Parameter Types:	HAL_BIT	HAL_FLOAT	HAL_S32	HAL_U32
Pin Directions:	HAL_IN	HAL_OUT	HAL_IO	
Parameter Directions:	HAL_RO	HAL_RW		

The full pin or parameter name is formed by joining the prefix and the suffix with a “.”, so in the example the pin created is called *passthrough.in*.

```
h.ready()
```

Once all the pins and parameters have been created, call the *.ready()* method.

15.15.3.1 Changing the prefix

The prefix can be changed by calling the `.setprefix()` method. The current prefix can be retrieved by calling the `.getprefix()` method.

15.15.4 Reading and writing pins and parameters

For pins and parameters which are also proper Python identifiers, the value may be accessed or set using the attribute syntax:

```
h.out = h.in
```

For all pins, whether or not they are also proper Python identifiers, the value may be accessed or set using the subscript syntax:

```
h['out'] = h['in']
```

To see all pins with their values, `getpins` returns all values in a dictionary of that component.

```
h.getpins()
>>>{'in': 0.0, 'out': 0.0}
```

15.15.4.1 Driving output (HAL_OUT) pins

Periodically, usually in response to a timer, all HAL_OUT pins should be "driven" by assigning them a new value. This should be done whether or not the value is different than the last one assigned. When a pin is connected to a signal, its old output value is not copied into the signal, so the proper value will only appear on the signal once the component assigns a new value.

15.15.4.2 Driving bidirectional (HAL_IO) pins

The above rule does not apply to bidirectional pins. Instead, a bidirectional pin should only be driven by the component when the component wishes to change the value. For instance, in the canonical encoder interface, the encoder component only sets the *index-enable* pin to **FALSE** (when an index pulse is seen and the old value is **TRUE**), but never sets it to **TRUE**. Repeatedly driving the pin **FALSE** might cause the other connected component to act as though another index pulse had been seen.

15.15.5 Exiting

A *halcmd unload* request for the component is delivered as a *KeyboardInterrupt* exception. When an unload request arrives, the process should either exit in a short time, or call the `.exit()` method on the component if substantial work (such as reading or writing files) must be done to complete the shutdown process.

15.15.6 Helpful Functions

15.15.6.1 ready

Tells the HAL system the component is initialized. Locks out adding pins.

15.15.6.2 **unready**

Allows a component to add pins after *ready()* has been called. One should call *ready()* on the component after.

15.15.6.3 **component_exists**

Does the specified component exist at this time.

Example

```
hal.component_exists("testpanel")
```

15.15.6.4 **component_is_ready**

Is the specified component ready at this time.

Example

```
hal.component_is_ready("testpanel")
```

15.15.6.5 **get_msg_level**

Get the current Realtime msg level.

15.15.6.6 **set_msg_level**

Set the current Realtime msg level. used for debugging information.

15.15.6.7 **connect**

Connect a pin to a signal.

Example

```
hal.connect("pinname","signal_name")
```

15.15.6.8 **disconnect**

Disconnect a pin from a signal.

Example

```
hal.disconnect("pinname")
```

15.15.6.9 **get_value**

Read a pin, param, or signal directly.

Example

```
value = hal.get_value("iocontrol.0.emc-enable-in")
```

15.15.6.10 get_info_pins()

Returns a list of dicts of all system pins.

```
listOfDicts = hal.get_info_pins()
pinName1 = listOfDicts[0].get('NAME')
pinValue1 = listOfDicts[0].get('VALUE')
pinType1 = listOfDicts[0].get('TYPE')
pinDirection1 = listOfDicts[0].get('DIRECTION')
```

15.15.6.11 get_info_signals()

Returns a list of dicts of all system signals.

```
listOfDicts = hal.get_info_signals()
signalName1 = listOfDicts[0].get('NAME')
signalValue1 = listOfDicts[0].get('VALUE')
driverPin1 = listOfDicts[0].get('DRIVER')
```

15.15.6.12 get_info_params()

Returns a list of dicts of all system parameters.

```
listOfDicts = hal.get_info_params()
paramName1 = listOfDicts[0].get('NAME')
paramValue1 = listOfDicts[0].get('VALUE')
paramDirection1 = listOfDicts[0].get('DIRECTION')
```

15.15.6.13 new_sig

Create a new signal of the type specified.

Example

```
hal.new_sig("signalname",hal.HAL_BIT)
```

15.15.6.14 pin_has_writer

Does the specified pin have a driving pin connected.
Returns True or False.

```
h.in.pin_has_writer()
```

15.15.6.15 get_name

Get the HAL object name.
Return a string.

```
h.in.get_name()
```

15.15.6.16 get_type

Get the HAL object's type.
Returns an integer.

```
h.in.get_type()
```

15.15.6.17 get_dir

Get the HAL object direction type.
Returns an integer.

```
h.in.get_dir()
```

15.15.6.18 get

Get the HAL object value.

```
h.in.get()
```

15.15.6.19 set

Set the HAL object value.

```
h.out.set(10)
```

15.15.6.20 is_pin

Is the object a pin or parameter?
Returns True or False.

```
h.in.is_pin()
```

15.15.6.21 sampler_base

TODO

15.15.6.22 stream_base

TODO

15.15.6.23 stream

TODO

15.15.6.24 set_p

Set a pin value of any pin in the HAL system.

Example

```
hal.set_p("pinname","10")
```

15.15.7 Constants

Use these to specify details rather than the value they hold.

- HAL_BIT
- HAL_FLOAT
- HAL_S32
- HAL_U32
- HAL_IN
- HAL_OUT
- HAL_RO
- HAL_RW
- MSG_NONE
- MSG_ALL
- MSG_DBG
- MSG_ERR
- MSG_INFO
- MSG_WARN

15.15.8 System Information

Read these to acquire information about the realtime system.

- is_kernelspace
 - is_rt
 - is_sim
 - is_userspace
-

15.15.9 Use with hal_glib in GladeVCP Handler

GladeVCP uses the hal_glib library, which can be used to connect a "watcher" signal on a HAL input pin.

This signal can be used to register a function to call when the HAL pin changes state.

One must import the module and the hal module:

```
import hal_glib
import hal
```

Then make a pin and connect a *value-changed* (the watcher) signal to a function call:

```
class HandlerClass:
    def __init__(self, halcomp, builder, useropts):
        self.example_trigger = hal_glib.GPin(halcomp.newpin('example-trigger', hal.HAL_BIT, ←
            hal.HAL_IN))
        self.example_trigger.connect('value-changed', self._on_example_trigger_change)
```

And have a function to be called:

```
def _on_example_trigger_change(self, pin, userdata=None):
    print("pin value changed to:" % (pin.get()))
    print("pin name= %s" % (pin.get_name()))
    print("pin type= %d" % (pin.get_type()))

    # this can be called outside the function
    self.example_trigger.get()
```

15.15.10 Use with hal_glib in QtVCP Handler

QtVCP uses the hal_glib library, which can be used to connect a "watcher" signal on a HAL input pin. This signal can be used to register a function to call when the HAL pin changes state.

One must import the hal module:

```
import hal
```

Then make a pin and connect a *value_changed* (the watcher) signal to a function call:

```
#####
# *** INITIALIZE *** #
#####
# widgets allows access to widgets from the qtvcp files
# at this point the widgets and HAL pins are not instantiated
def __init__(self, halcomp, widgets, paths):
    self.hal = halcomp
    self.testPin = self.hal.newpin('test-pin', hal.HAL_BIT, hal.HAL_IN)
    self.testPin.value_changed.connect(lambda s: self.setTestPin(s))
```

And have a function to be called. This shows ways to get the pin value and information.

```
#####
# general functions #
#####
def setTestPin(self, data):
    print("Test pin value changed to:" % (data))
    print('halpin object =', self.w.sender())
```

```
print('halpin name: ',self.sender().text())
print('halpin type: ',self.sender().get_type())

# this can be called outside the function
print(self.testPin.get())
```

15.15.11 Project ideas

- Create an external control panel with buttons, switches, and indicators. Connect everything to a microcontroller, and connect the microcontroller to the PC using a serial interface. Python has a very capable serial interface module called [pyserial](#) (Ubuntu package name "python3-serial", in the universe repository)
- Attach a [LCDProc](#)-compatible LCD module and use it to display a digital readout with information of your choice (Ubuntu package name "lcdproc", in the universe repository)
- Create a virtual control panel using any GUI library supported by Python (gtk, Qt, wxWindows, etc.)

Part V

Advanced Topics

Chapter 16

Kinematics

16.1 Introduction

When we talk about CNC machines, we usually think about machines that are commanded to move to certain locations and perform various tasks. In order to have an unified view of the machine space, and to make it fit the human point of view over 3D space, most of the machines (if not all) use a common coordinate system called the Cartesian Coordinate System.

The Cartesian Coordinate system is composed of three axes (X, Y, Z) each perpendicular to the other two ¹.

When we talk about a G-code program (RS274/NGC) we talk about a number of commands (G0, G1, etc.) which have positions as parameters (X- Y- Z-). These positions refer exactly to Cartesian positions. Part of the LinuxCNC motion controller is responsible for translating those positions into positions which correspond to the machine kinematics ².

16.1.1 Joints vs Axes

A joint of a CNC machine is a one of the physical degrees of freedom of the machine. This might be linear (leadscrews) or rotary (rotary tables, robot arm joints). There can be any number of joints on a given machine. For example, one popular robot has 6 joints, and a typical simple milling machine has only 3.

There are certain machines where the joints are laid out to match kinematics axes (joint 0 along axis X, joint 1 along axis Y, joint 2 along axis Z), and these machines are called Cartesian machines (or machines with Trivial Kinematics). These are the most common machines used in milling, but are not very common in other domains of machine control (e.g. welding: puma-typed robots).

LinuxCNC supports axes with names: X Y Z A B C U V W. The X Y Z axes typically refer to the usual Cartesian coordinates. The A B C axes refer to rotational coordinates about the X Y Z axes respectively. The U V W axes refer to additional coordinates that are commonly made colinear to the X Y Z axes respectively.

16.2 Trivial Kinematics

The simplest machines are those in which each joint is placed along one of the Cartesian axes. On these machines the mapping from Cartesian space (the G-code program) to the joint space (the

¹The word "axes" is also commonly (and wrongly) used when talking about CNC machines, and referring to the moving directions of the machine.

²Kinematics: a two way function to transform from Cartesian space to joint space.

actual actuators of the machine) is trivial. It is a simple 1:1 mapping:

```
pos->tran.x = joints[0];
pos->tran.y = joints[1];
pos->tran.z = joints[2];
```

In the above code snippet one can see how the mapping is done: the X position is identical with the joint 0, the Y position with joint 1, etc. The above refers to the direct kinematics (one direction of the transformation). The next code snippet refers to the inverse kinematics (or the inverse direction of the transformation):

```
joints[0] = pos->tran.x;
joints[1] = pos->tran.y;
joints[2] = pos->tran.z;
```

In LinuxCNC, the identity kinematics are implemented with the *trivkins* kinematics module and extended to 9 axes. The default relationships between axis coordinates and joint numbers are: ^{3 4}

```
pos->tran.x = joints[0];
pos->tran.y = joints[1];
pos->tran.z = joints[2];
pos->a      = joints[3];
pos->b      = joints[4];
pos->c      = joints[5];
pos->u      = joints[6];
pos->v      = joints[7];
pos->w      = joints[8];
```

Similarly, the default relationships for inverse kinematics for *trivkins* are:

```
joints[0] = pos->tran.x;
joints[1] = pos->tran.y;
joints[2] = pos->tran.z;
joints[3] = pos->a;
joints[4] = pos->b;
joints[5] = pos->c;
joints[6] = pos->u;
joints[7] = pos->v;
joints[8] = pos->w;
```

It is straightforward to do the transformation for a trivial "kins" (*trivkins* kinematics) or Cartesian machine provided that there are no omissions in the axis letters used.

It gets a bit more complicated if the machine is missing one or more of the axis letters. The problems of omitted axis letters is addressed by using the *coordinates=* module parameter with the *trivkins* module. Joint numbers are assigned consecutively to each coordinate specified. A lathe can be described with *coordinates=xz*. The joint assignments will then be:

```
joints[0] = pos->tran.x
joints[1] = pos->tran.z
```

Use of the *coordinates=* parameter is recommended for configurations that omit axis letters. ⁵

³If the machine (for example a lathe) is mounted with only the X, Z and A axes and the INI file of LinuxCNC contains only the definition of these 3 joints, then the previous assertion is false. Because we currently have (joint0=X, joint1=Z, joint2=A) which assumes that joint1=Y. To make this work in LinuxCNC just define all the axes (XYZA), LinuxCNC will then use a simple loop in HAL for unused Y axis.

⁴Another way to make it work is to change the corresponding code and recompile the software.

⁵Historically, the *trivkins* module did not support the *coordinates=* parameter so lathe configs were often configured as XYZ machines. The unused Y axis was configured to 1) home immediately, 2) use a simple loopback to connect its position command HAL pin to its position feedback HAL pin, and 3) hidden in gui displays. Numerous sim configs use these methods in order to share common HAL files.

The *trivkins* kinematics module also allows the same coordinate to be specified for more than one joint. This feature can be useful on machines like a gantry having two independent motors for the y coordinate. Such a machine could use *coordinates=xyyz* resulting in joint assignments:

```
joints[0] = pos->tran.x  
joints[1] = pos->tran.y  
joints[2] = pos->tran.y  
joints[3] = pos->tran.z
```

See the trivkins man pages for more information.

16.3 Non-trivial kinematics

There can be quite a few types of machine setups (robots: puma, scara; hexapods etc.). Each of them is set up using linear and rotary joints. These joints don't usually match with the Cartesian coordinates, therefore we need a kinematics function which does the conversion (actually 2 functions: forward and inverse kinematics function).

To illustrate the above, we will analyze a simple kinematics called bipod (a simplified version of the tripod, which is a simplified version of the hexapod).

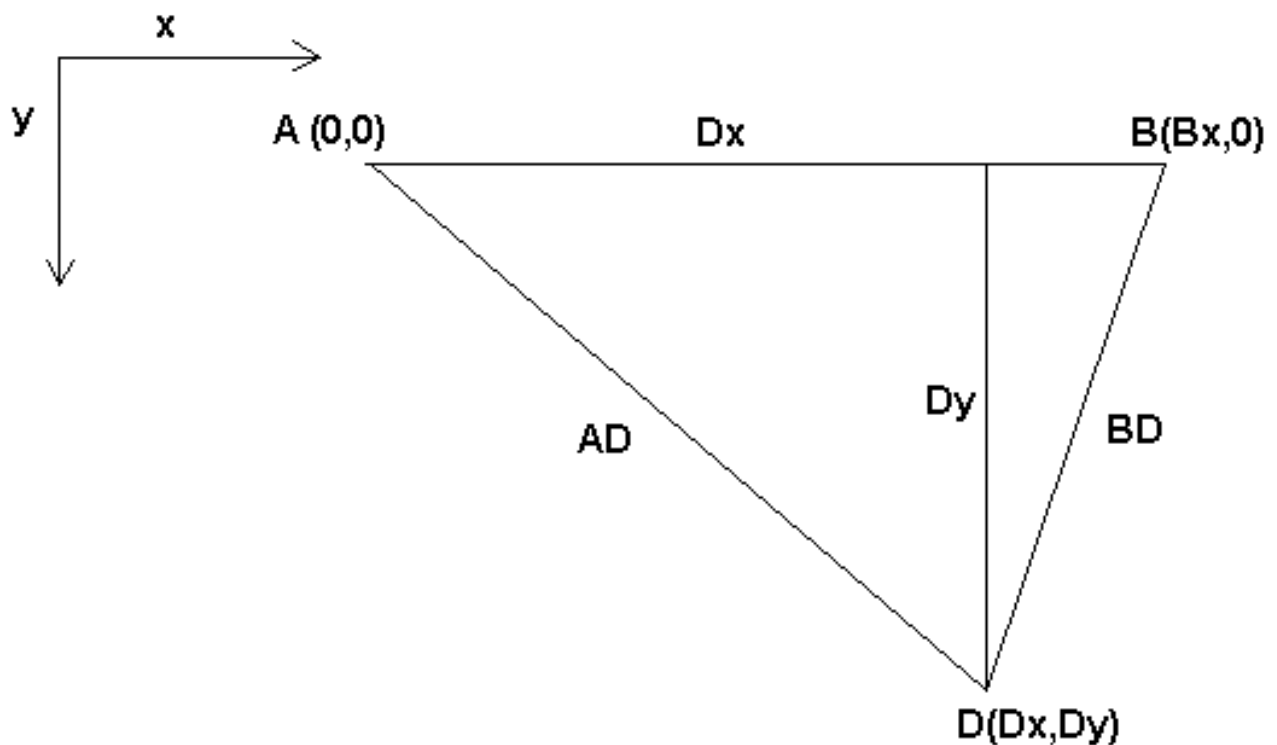


Figure 16.1: Bipod setup

The Bipod we are talking about is a device that consists of 2 motors placed on a wall, from which a device is hung using some wire. The joints in this case are the distances from the motors to the device (named AD and BD in the figure).

The position of the motors is fixed by convention. Motor A is in (0,0), which means that its X coordinate is 0, and its Y coordinate is also 0. Motor B is placed in (Bx, 0), which means that its X coordinate is Bx.

Our tooltip will be in point D which gets defined by the distances AD and BD, and by the Cartesian coordinates Dx, Dy.

The job of the kinematics is to transform from joint lengths (AD, BD) to Cartesian coordinates (Dx, Dy) and vice-versa.

16.3.1 Forward transformation

To transform from joint space into Cartesian space we will use some trigonometry rules (the right triangles determined by the points (0,0), (Dx,0), (Dx,Dy) and the triangle (Dx,0), (Bx,0) and (Dx,Dy)).

We can easily see that:

$$AD^2 = x^2 + y^2 \quad BD^2 = (Bx - x)^2 + y^2$$

likewise:

$$BD^2 = (Bx - x)^2 + y^2$$

If we subtract one from the other we will get:

$$AD^2 - BD^2 = x^2 + y^2 - x^2 + 2 * x * Bx - Bx^2 - y^2$$

and therefore:

$$x = \frac{AD^2 - BD^2 + Bx^2}{2 * Bx}$$

From there we calculate:

$$y = \sqrt{AD^2 - x^2}$$

Note that the calculation for y involves the square root of a difference, which may not result in a real number. If there is no single Cartesian coordinate for this joint position, then the position is said to be a singularity. In this case, the forward kinematics return -1.

Translated to actual code:

```
double AD2 = joints[0] * joints[0];
double BD2 = joints[1] * joints[1];
double x = (AD2 - BD2 + Bx * Bx) / (2 * Bx);
double y2 = AD2 - x * x;
if(y2 < 0) return -1;
pos->tran.x = x;
pos->tran.y = sqrt(y2);
return 0;
```


16.3.2 Inverse transformation

The inverse kinematics is much easier in our example, as we can write it directly:

$$AD = \sqrt{x^2 + y^2}$$

$$BD = \sqrt{(Bx - x)^2 + y^2}$$

or translated to actual code:

```
double x2 = pos->tran.x * pos->tran.x;
double y2 = pos->tran.y * pos->tran.y;
joints[0] = sqrt(x2 + y2);
joints[1] = sqrt((Bx - pos->tran.x)*(Bx - pos->tran.x) + y2);
return 0;
```

16.4 Implementation details

A kinematics module is implemented as a HAL component, and is permitted to export pins and parameters. It consists of several "C" functions (as opposed to HAL functions):

```
int kinematicsForward(const double *joint, EmcPose *world,
const KINEMATICS_FORWARD_FLAGS *fflags,
KINEMATICS_INVERSE_FLAGS *iflags)
```

Implements the [forward kinematics function](#).

```
int kinematicsInverse(const EmcPose * world, double *joints,
const KINEMATICS_INVERSE_FLAGS *iflags,
KINEMATICS_FORWARD_FLAGS *fflags)
```

Implements the inverse kinematics function.

```
KINEMATICS_TYPE kinematicsType(void)
```

Returns the kinematics type identifier, típicamente *KINEMATICS_BOTH*:

1. KINEMATICS_IDENTITY (each joint number corresponds to an axis letter)
2. KINEMATICS_BOTH (forward and inverse kinematics functions are provided)
3. KINEMATICS_FORWARD_ONLY
4. KINEMATICS_INVERSE_ONLY

Note

GUIs may interpret KINEMATICS_IDENTITY to hide the distinctions between joint numbers and axis letters when in joint mode (typically prior to homing).

```
int kinematicsSwitchable(void)
int kinematicsSwitch(int switchkins_type)
KINS_NOT_SWITCHABLE
```

The function `kinematicsSwitchable()` returns 1 if multiple kinematics types are supported. The function `kinematicsSwitch()` selects the kinematics type. See [Switchable Kinematics](#).

Note

The majority of provided kinematics modules support a single kinematics type and use the directive **"KINS_NOT_SWITCHABLE"** to supply defaults for the required `kinematicsSwitchable()` and `kinematicsSwitch()` functions.

```
int kinematicsHome(EmcPose *world, double *joint,
KINEMATICS_FORWARD_FLAGS *fflags,
KINEMATICS_INVERSE_FLAGS *iflags)
```

The home kinematics function sets all its arguments to their proper values at the known home position. When called, these should be set, when known, to initial values, e.g., from an INI file. If the home kinematics can accept arbitrary starting points, these initial values should be used.

```
int rtapi_app_main(void)
void rtapi_app_exit(void)
```

These are the standard setup and tear-down functions of RTAPI modules.

When they are contained in a single source file, kinematics modules may be compiled and installed by *halcompile*. See the *halcompile(1)* manpage or the HAL manual for more information.

16.4.1 Kinematics module using the `userkins.comp` template

Another way to create a custom kinematics module is to adapt the HAL component *userkins*. This template component can be modified locally by a user and can be built using *halcompile*.

See the *userkins* man pages for more information.

Note that to create switchable kinematic modules the required modifications are somewhat more complicated.

See *millturn.comp* as an example of a switchable kinematic module that was created using the *userkins.comp* template.

Chapter 17

Setting up "modified" Denavit-Hartenberg (DH) parameters for *genserkins*

17.1 Prelude

LinuxCNC supports a number of kinematics modules including one that supports a generalized set of serial kinematics commonly specified via Denavit-Hartenberg parameters.

This document illustrates a method to set up the DH-parameters for a Mitsubishi RV-6SDL in LinuxCNC using *genserkins* kinematics.

Note

This document does not cover the creation of a *vismach* model which, while certainly very useful, requires just as much careful modeling if it is to match the *genserkins* model derived in this document.

Note

There may be errors and/or shortcomings — use at your own risk!

17.2 General

With the proliferation of industrial robots comes an increased interest to control used robots with LinuxCNC. A common type of robot used in industry and manufacturing is the "serial manipulator" designed as a series of motorized joints connected by rigid links. Serial robots often have six joints as required for the six degrees of freedom needed to both position (XYZ) and orient (ABC or pitch, roll, yaw) an object in space. Often these robots have an arm structure that extends from a base to an end-effector.

Control of such a serial robot requires the calculation of the end-effector's position and orientation in relation to a reference coordinate system when the joint angles are known (**forward kinematics**) and also the more complex reverse calculation of the required joint angles for a given end-effector position and orientation in relation to the reference coordinate system (**inverse kinematics**). The standard mathematical tools used for these calculations are matrices which are basically tables of parameters and formulas that make it easier to handle the rotations and translations involved in forward and inverse kinematics calculations.

Detailed familiarity with the math is not required for a serial robot since LinuxCNC provides a kinematics module that implements an algorithm called *genserkins* to calculate the forward and inverse kinematics for a generic serial robot. In order to control a specific serial robot, *genserkins* must be provided with data so that it can build a mathematical model of the robot's mechanical structure and thus do the math.

The required data needs to be in a standardized form that has been introduced by Jacques Denavit and Richard Hartenberg back in the fifties and are called the DH-Parameters. Denavit and Hartenberg used four parameters to describe how one joint is linked to the next. These parameters describe basically two rotations (*alpha* and *theta*) and two translations (*a* and *d*).

17.3 Modified DH-Parameters

As is often the case, this "standard" has been modified by other authors who have introduced "modified DH-parameters" and one must be very careful because *genserkins* uses "modified DH-parameters" as described in the publication "Introduction to Robotics, Mechanics and Control" by John J. Craig. Beware there is a lot of information to be found on *DH-parameters* but rarely does the author define which convention is actually used. In addition, some people have found it necessary to change the parameter named *a* to *r* and have thus added to the confusion. This document adheres to the convention in the above mentioned publication by Craig with the difference that joint and parameter enumeration begins with the number 0 in order to be consistent with *genserkins* and its HAL pins.

Standard and Modified DH-Parameters consist of four numeric values for each joint (*a*, *d*, *alpha* and *theta*) that describe how the coordinate system (CS) sitting in one joint has to be moved and rotated to be aligned with the next joint. Aligned means that the Z-axis of our CS coincides with the axis of rotation of the joint and points in the positive direction such that, using the right hand rule with the thumb pointing in the positive direction of the Z-axis, the fingers point in the positive direction of rotation of the joint. It becomes clear that in order to do this, one must decide on the positive directions of all joints before starting to derive the parameters!

The difference between "standard" and "modified" notations is in how the parameters are allocated to the links. Using the "standard" DH-Parameters in *genserkins* will **not** give the correct mathematical model.

17.4 Modified DH-Parameters as used in *genserkins*

Note that *genserkins* does not handle offsets to theta-values — theta is the joint variable that is **controlled** by LinuxCNC. With the CS aligned with the joint, a rotation around its Z-Axis is identical to the rotation commanded to that joint by LinuxCNC. This makes it impossible to define the 0° position of our robots joints arbitrarily.

The three configurable parameters are:

1. **alpha** : positive or negative rotation (in radians) around the X-axis of the "current coordinate system"
2. **a** : positive distance, along X, between two joint axes specified in *machine units* (mm or inch) defined in the system's INI file.
3. **d** : positive or negative length along Z (also in *machine units*)

The parameter sets are always derived in the same order and a set is completed by setting the **d**-parameter. This does not leave the Z-axis of our CS aligned with the next joint! This may seem confusing but sticking to this rule will yield a working set of parameters. Once the **d**-parameter is set, the X-axis of our-CS needs to point to the axis of the next joint.

17.5 Numbering of joints and parameters

The first joint in LinuxCNC is joint-0 (because in software counting starts with 0) while most publications start with the number 1. That goes for all the parameters as well. That is, numbering starts with a-0, alpha-0, d-0 and ends with a-5, alpha-5 and d-5. Keep this in mind when following a publication to set up *genserkins* parameters.

17.6 How to start

Convention is to start by placing the reference-CS in the base of the robot with it's Z-axis coinciding with the axis of the first joint and its X-axis pointing toward the next joint's axis.

This will also result in the DRO values in LinuxCNC being referenced to that point. Having done so sets a-0 and alpha-0 to 0. The above mentioned publication (Craig) also sets d-0 to 0, which is confusing when a displacement offset is needed in order to have the reference-CS at the bottom of the base. Setting d-0 = to the displacement gives correct results. In this manner, the first set of parameters are alpha-0 = 0, a-0 = 0, d0 = displacement, and the X-axis of the CS points to the axis of the next joint (joint-1).

Derivation of the net set (alpha-1, a-1, d-1) follows — always using the same sequence all the way to the sixth set (alpha-5, a-5, d-5).

And thus, the TCP-CS of the end-effector is sitting in the center of the hand flange.

17.7 Special cases

If the next joint-axis is parallel to the last then one could arbitrarily choose a value for the d-parameter but there is no point in setting it other than 0.

17.8 Detailed Example (RV-6SL)

Described below is a method to derive the required "modified DH-parameters" for a Mitsubishi RV-6SDL and how to set the parameters in the HAL file to be used with the *genserkins* kinematics in LinuxCNC. The necessary dimensions are best taken from a dimensional drawing provided by the manufacturer of the robot.

A-0, ALPHA-0

We choose our base coordinate system at the intersection of the axis of joint-0 and the base plate. We point the X-axis towards the end effector and the Z-axis pointing up. Note that the rotation direction of joint-0 is right handed to our Z-axis. Also note that because the Z-axis of our coordinate system coincides with the axis of joint-0 and points in the same direction alpha-0 and a-0 are 0.

We set:

```
setp genserkins.A-0 = 0
```

```
setp genserkins.ALPHA-0 = 0
```

Axis of Joint 0

Positive rotation of Joint 0

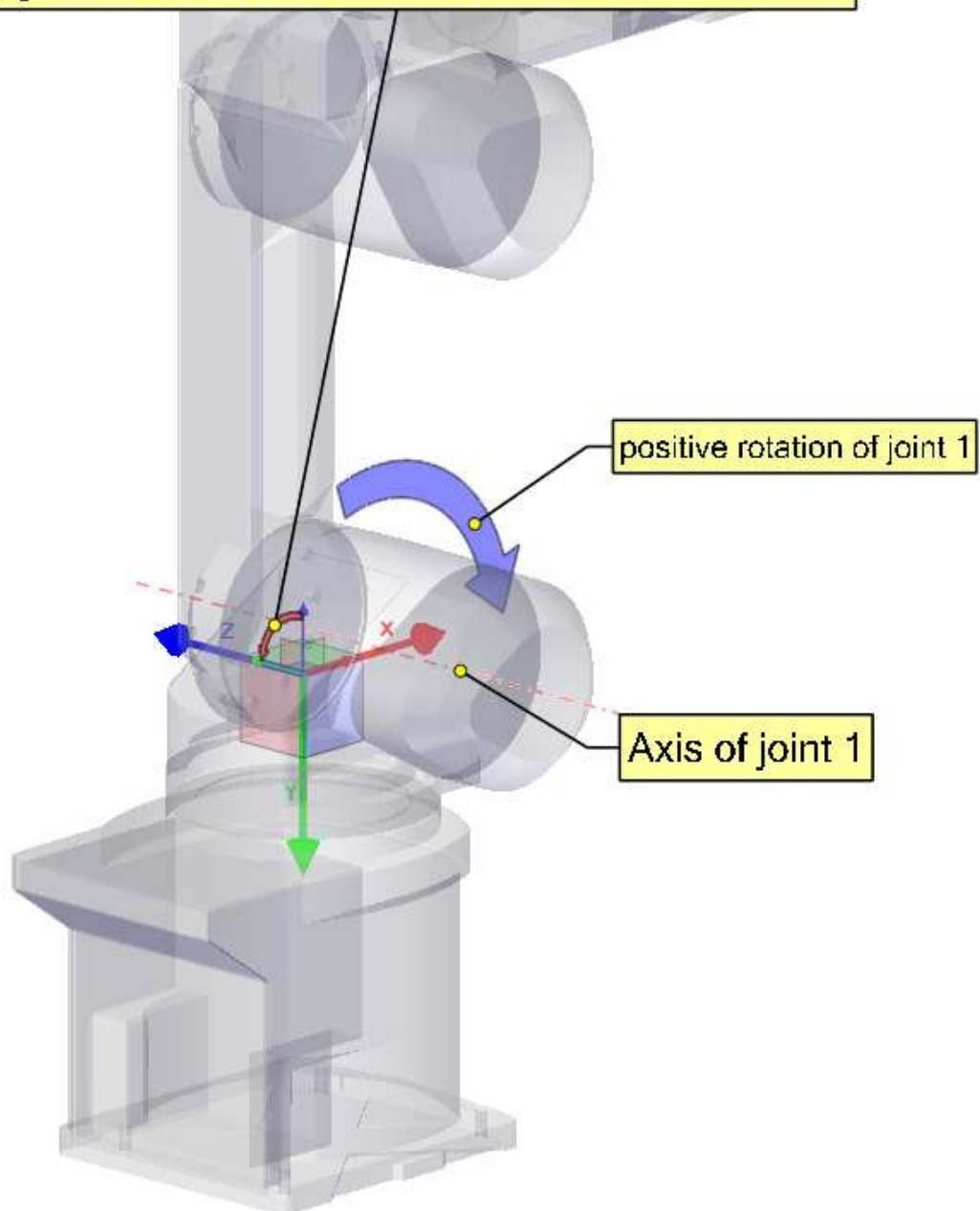




ALPHA-1

To make our Z-axis face the same direction as the axis of joint-1 we need to rotate our coordinate system 90° around its X-axis in the negative sense (use right hand rule with thumb along X). A rotation around X corresponds to an alpha-value. Note the alpha values have to be defined in radians. As 360° is equal to 2π our -90° is equal to $-\pi/2 = -1.570796327$

```
setp genserkins.ALPHA-1 = -1.570796327
```



A-1

To make our Z-axis colinear with the axis of joint-1 we need to move our coordinate system 85mm along the X-axis.

```
setp genserkins.A-1 = 85
```



**Note:**

In order to make our X-axis intersect the axis of joint-2 we would need to rotate our coordinate system around its Z-axis. To do this we could, in theory, define theta-1 equal to -90° .

However gensekkins does not allow the definition of theta values. In gensekkins.c we see that the theta values for all joints are set to 0.

Now theta of course is the rotation of the joint itself and so is variable in an angular joint. Theta values are only used to define the home pose of a robot in the way of an offset.

So if we could define theta-1 equal to -90° we could define joint-1 to be oriented this way for 0° . Since we cannot define it we need to rotate joint-1 in a way so that our X-axis intersects with the axis of the next joint.



By rotating our joint-1 by 90° we made our X-axis intersect the axis of the next joint and we can continue defining our DH-Parameters.

D-1

Since, after we rotated joint 1, our X-axis already intersects the axis of joint-2 we do not need to move our coordinate system along the Z-axis.

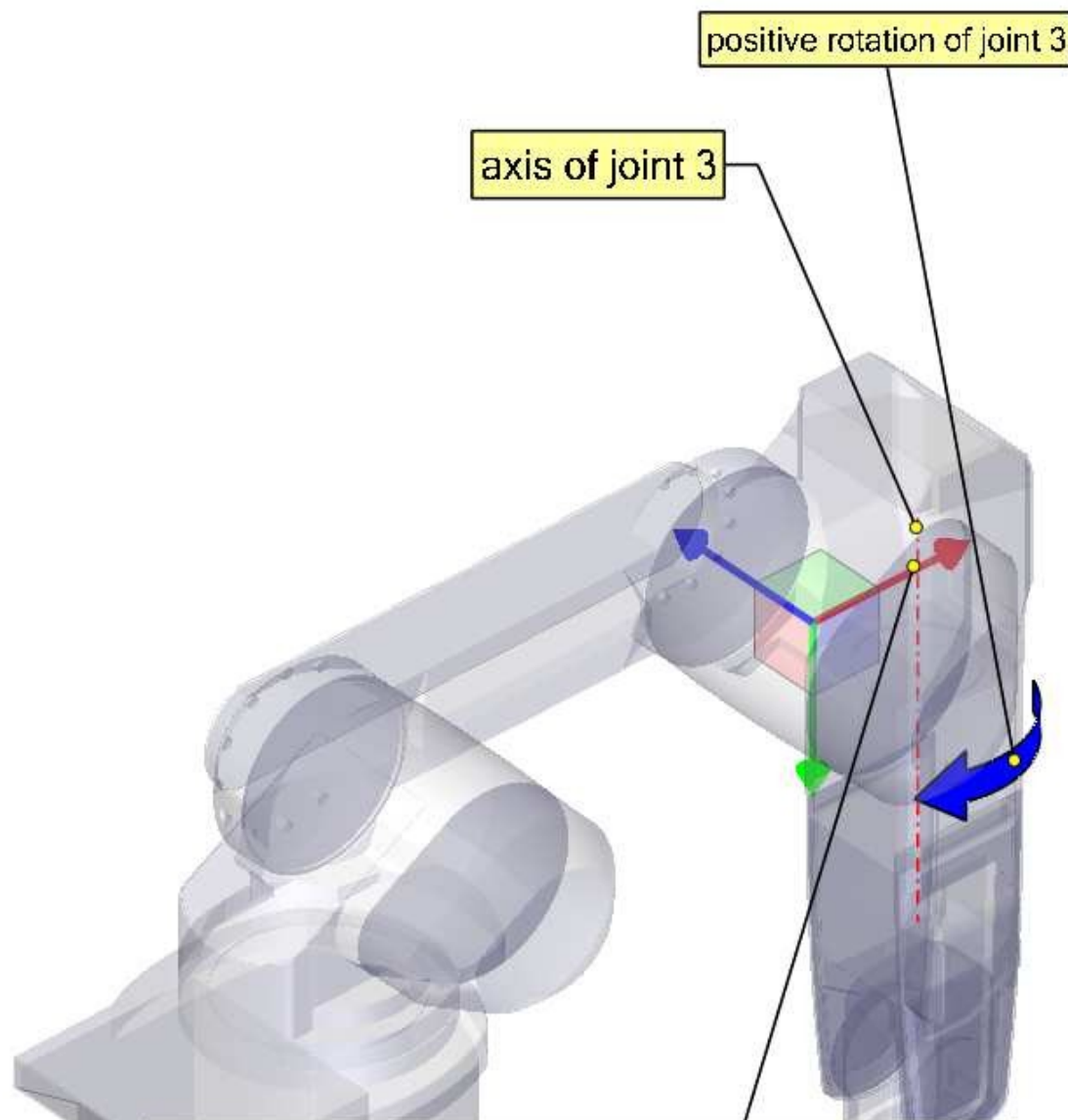
```
setp genserkins.D-1 = 0
```

ALPHA-2

The axis of joint -2 is parallel to the axis of joint -1 and points in the same direction. Thus we do not need to rotate our Z-axis.

```
setp genserkins.ALPHA-2 = 0
```





D-2

Our X-axis again already intersects the axis of the next joint 3. So our d2 parameter is again 0.

```
setp genserkins.D-2 = 0
```

ALPHA-3

To make our Z-axis face the same direction as the axis of joint-3 we need to rotate or coordinate system 90° around its X-axis in the negative sense (use right hand rule with thumb along X). A rotation around X corresponds to an alpha-value. Note the alpha values have to be defined in radians. As 360° is equal to 2π our -90° is equal to $-\pi/2 = -1.570796327$

```
setp genserkins.ALPHA-3 = -1.570796327
```


After rotating our coordinate system by ALPHA-3 our Z-axis points in the same direction as the axis of joint 3.

Our modified DH-Parameters so far:

ALPHA-0 = 0
ALPHA-1 = -1.570796327
ALPHA-2 = 0
ALPHA-3 = -1.570796327

A-0 = 0
A-1 = 85
A-2 = 380

D-0 = 350
D-1 = 0
D-2 = 0



A-3

To make our Z-axis colinear with the axis of joint 3 we need to move our coordinate system 100mm along its X-axis.

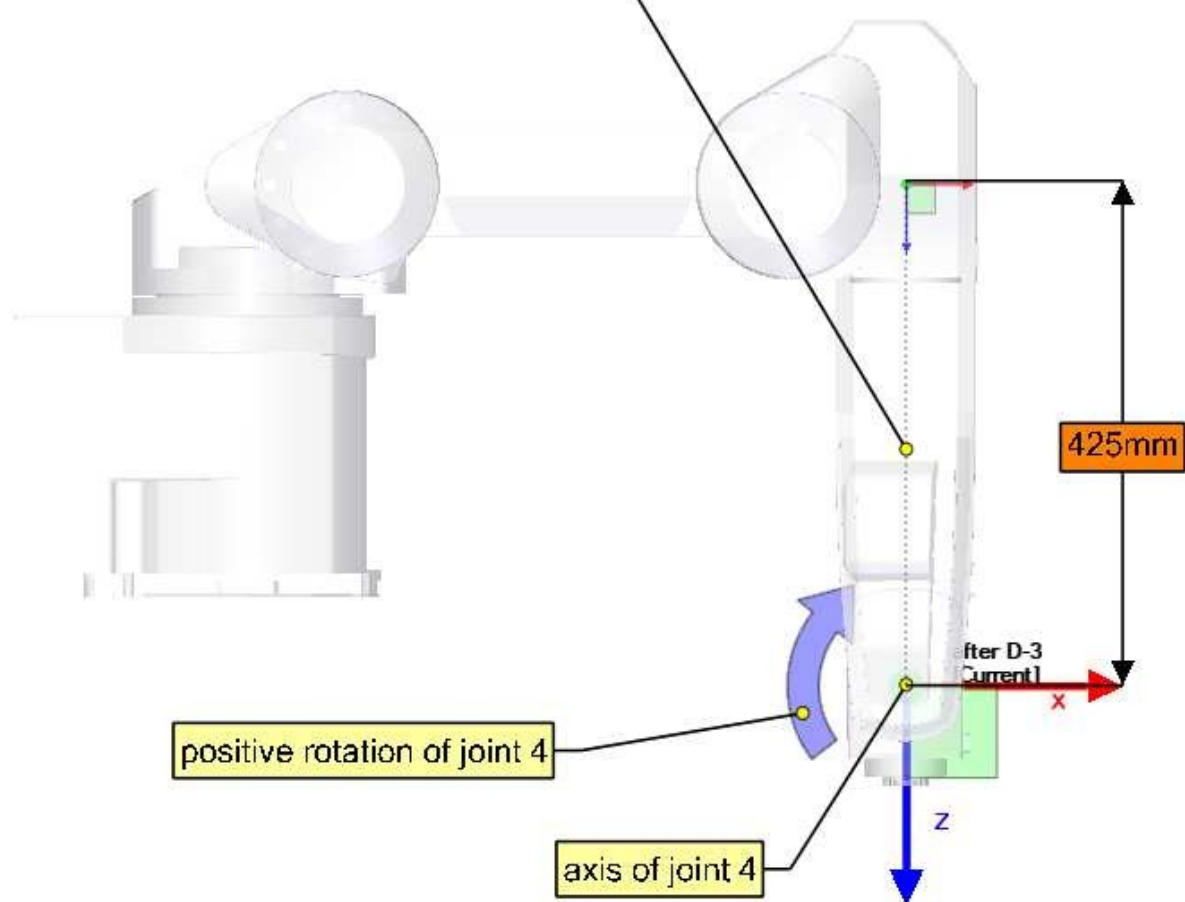
```
setp genserkins.A-3 = 100
```



D-3

We move our coordinate system 425 mm along its Z-Axis until its X-Axis intersects the axis of joint 4.

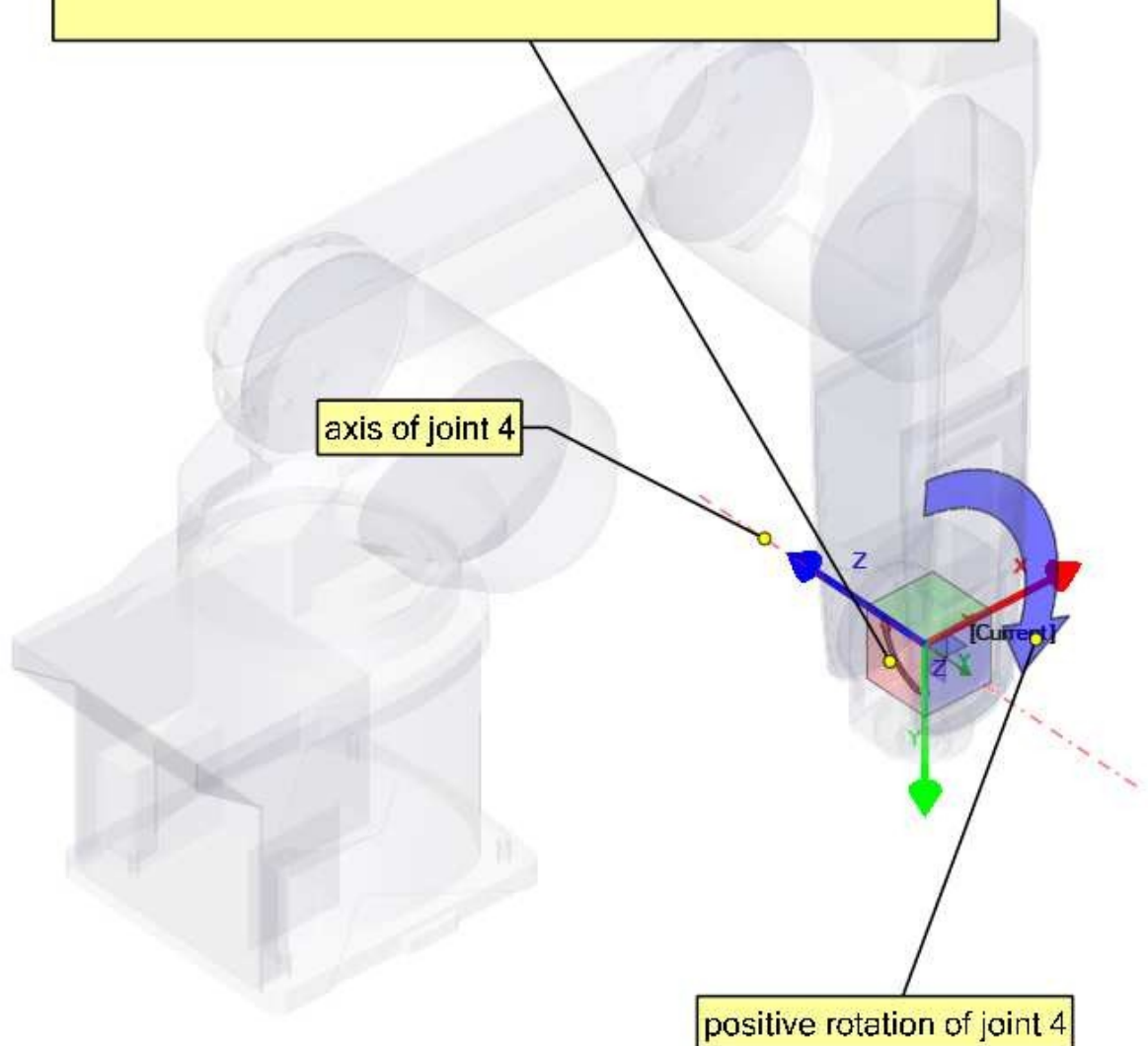
setp genserkins.D-3 = 425



ALPHA-4

To make our Z-axis face the same direction as the axis of joint-4 we need to rotate our coordinate system 90° around its X-axis in the positive sense (use right hand rule with thumb along X). A rotation around X corresponds to an alpha-value. Note the alpha values have to be defined in radians. As 360° is equal to 2π our 90° is equal to $\pi/2 = 1.570796327$

```
setp genserkins.ALPHA-4 = 1.570796327
```



Our modified DH-Parameters so far:

ALPHA-0 = 0
ALPHA-1 = -1.570796327
ALPHA-2 = 0
ALPHA-3 = -1.570796327
ALPHA-4 = 1.570796327

A-0 = 0
A-1 = 85
A-2 = 380
A-3 = 100

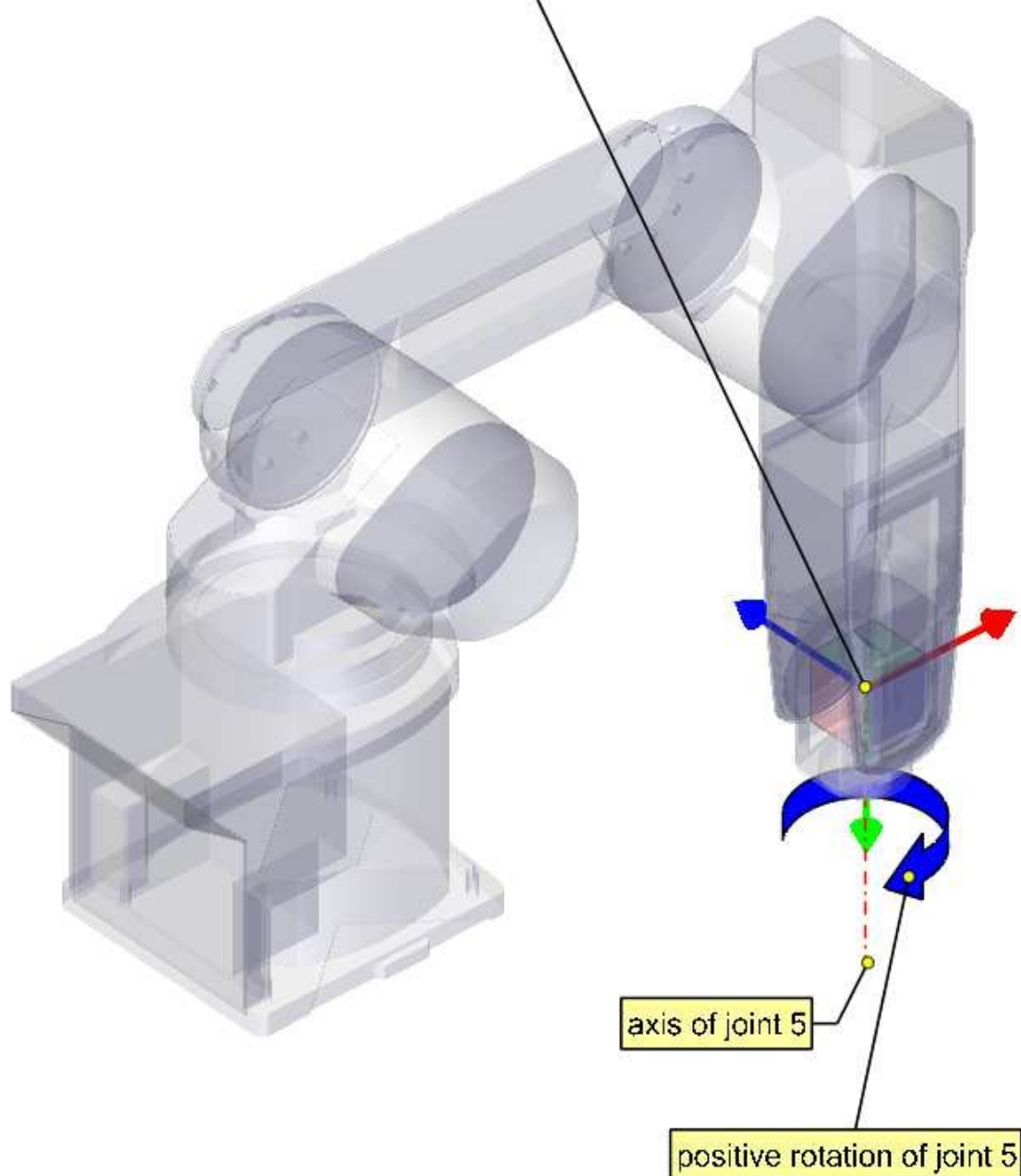
D-0 = 350
D-1 = 0
D-2 = 0
D-3 = 425



A-4

Since the origin of our coordinate system intersects the axis of the next joint-5 we can set A-4 to 0.

```
setp genserkins.A-4 = 0
```



D-4

Since the origin of our coordinate system lies on the axis of the next joint our d4 parameter is also 0.

```
setp genserkins.D-4 = 0
```

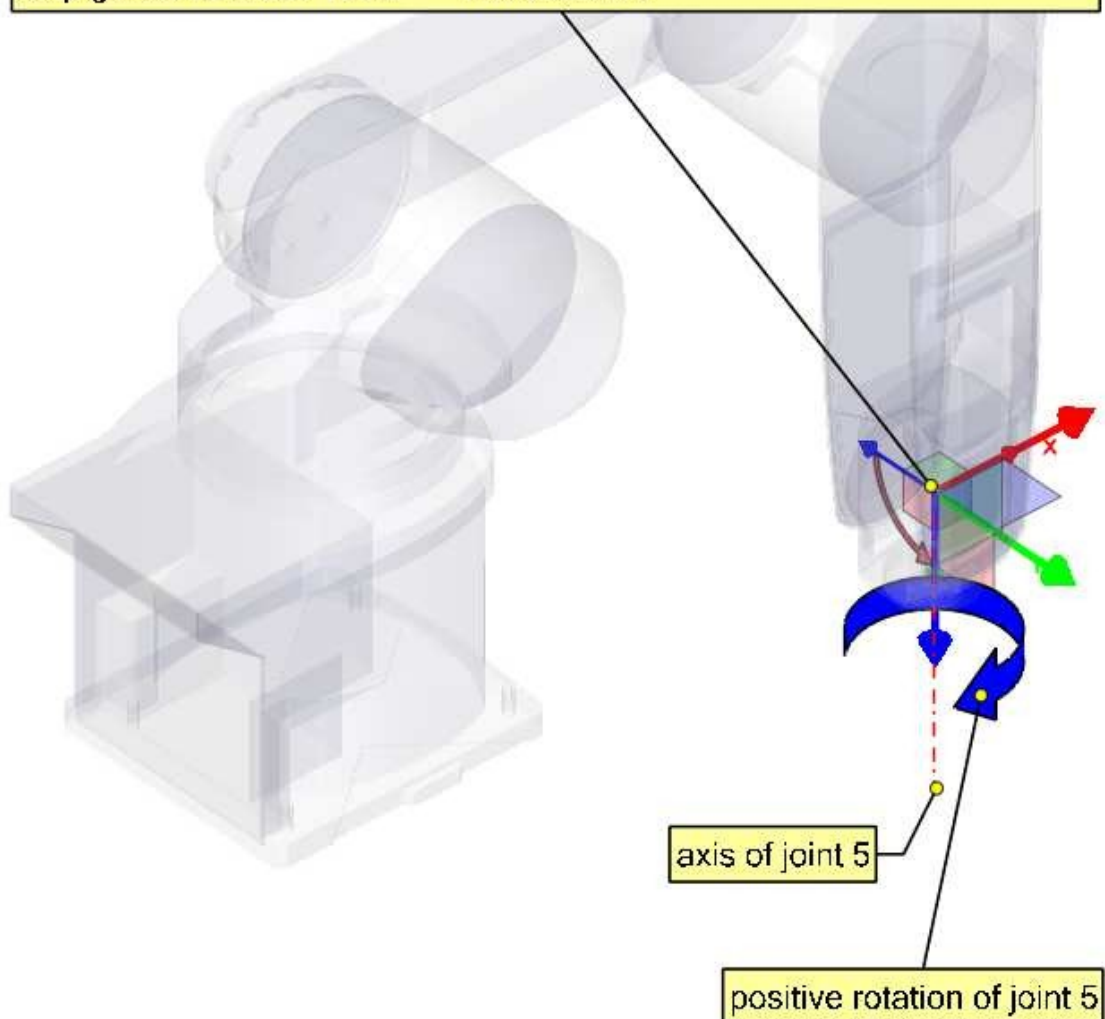
ALPHA-5

To make our Z-axis face the same direction as the axis of joint-5 we need to rotate our coordinate system 90° around its X-axis in the negative sense (use right hand rule with thumb along X).

A rotation around X corresponds to an alpha-value.

Note the alpha values have to be defined in radians. As 360° is equal to 2π our -90° is equal to $-\pi/2 = -1.570796327$

```
setp genserkins.ALPHA-5 = -1.570796327
```

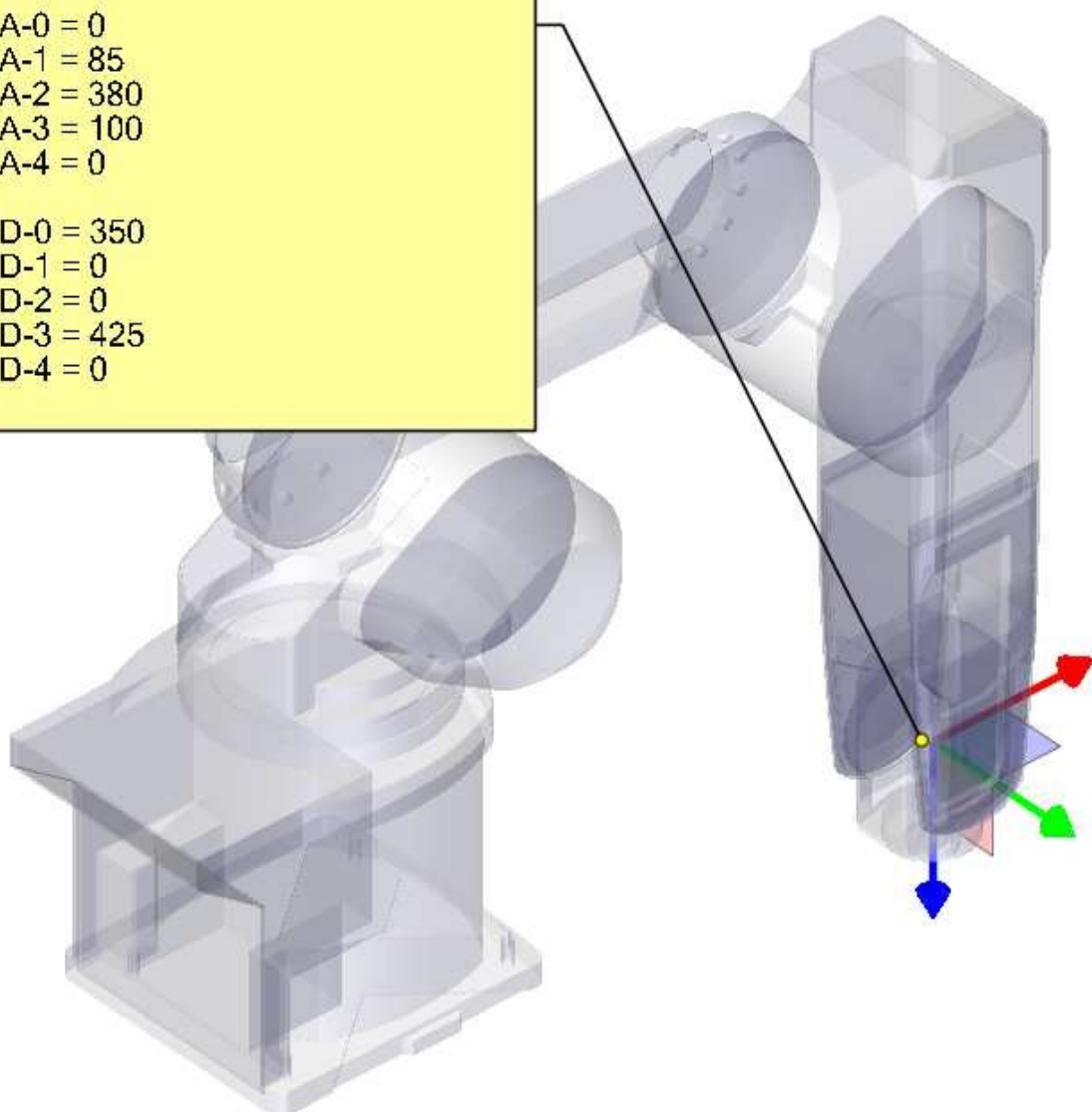


Our modified DH-Parameters so far:

ALPHA-0 = 0
ALPHA-1 = -1.570796327
ALPHA-2 = 0
ALPHA-3 = -1.570796327
ALPHA-4 = 1.570796327
ALPHA-5 = -1.570796327

A-0 = 0
A-1 = 85
A-2 = 380
A-3 = 100
A-4 = 0

D-0 = 350
D-1 = 0
D-2 = 0
D-3 = 425
D-4 = 0



D-5

We move our coordinate system for 85mm along its Z-axis.

With this we have finished setting up our modified DH- parameters and leaves our coordinate system at the center of the hand flange.

```
setp genserkins.D-5 = 85
```



Our final modified DH-Parameters:

ALPHA-0 = 0
ALPHA-1 = -1.570796327
ALPHA-2 = 0
ALPHA-3 = -1.570796327
ALPHA-4 = 1.570796327
ALPHA-5 = -1.570796327

A-0 = 0
A-1 = 85
A-2 = 380
A-3 = 100
A-4 = 0
A-5 = 0

D-0 = 350
D-1 = 0
D-2 = 0
D-3 = 425
D-4 = 0
D-5 = 85



17.9 Credits

Thanks to user Aciera for all text and the graphics for the RV-6SL robot!

Chapter 18

5-Axis Kinematics

18.1 Introduction

Coordinated multi-axis CNC machine tools controlled with LinuxCNC, require a special kinematics component for each type of machine. This chapter describes some of the most popular 5-axis machine configurations and then develops the forward (from work to joint coordinates) and inverse (from joint to work) transformations in a general mathematical process for two types of machine.

The kinematics components are given as well as vismach simulation models to demonstrate their behaviour on a computer screen. Examples of HAL file data are also given.

18.2 5-Axis Machine Tool Configurations

In this section we deal with the typical 5-axis milling or router machines with five joints or degrees-of-freedom which are controlled in coordinated moves.

3-axis machine tools cannot change the tool orientation, so 5-axis machine tools use two extra axes to set the cutting tool in an appropriate orientation for efficient machining of freeform surfaces.

Typical 5-axis machine tool configurations are shown in Figs. 3, 5, 7 and 9-11 [1,2] in section Figures.

The kinematics of 5-axes machine tools are much simpler than that of 6-axis serial arm robots, since 3 of the axes are normally linear axes and only two are rotational axes.

18.3 Tool Orientation and Location

CAD/CAM systems are typically used to generate the 3D CAD models of the workpiece as well as the CAM data for input to the CNC 5-axis machine. The tool or cutter location (CL) data, is composed of the cutter tip position and the cutter orientation relative to the workpiece coordinate system. Two vectors, as generated by most CAM systems and shown in Fig. 1, contain this information:

$$K = \begin{bmatrix} K_x \\ K_y \\ K_z \\ 0 \end{bmatrix} \quad \text{orientation vector;} \quad Q = \begin{bmatrix} Q_x \\ Q_y \\ Q_z \\ 1 \end{bmatrix} \quad \text{position vector} \quad (1)$$

The K vector is equivalent to the 3rd vector from the pose matrix E_6 that was used in the 6-axis robot kinematics [3] and the Q vector is equivalent to the 4th vector of E_6 . In MASTERCAM for example this information is contained in the intermediate output ".nci" file.

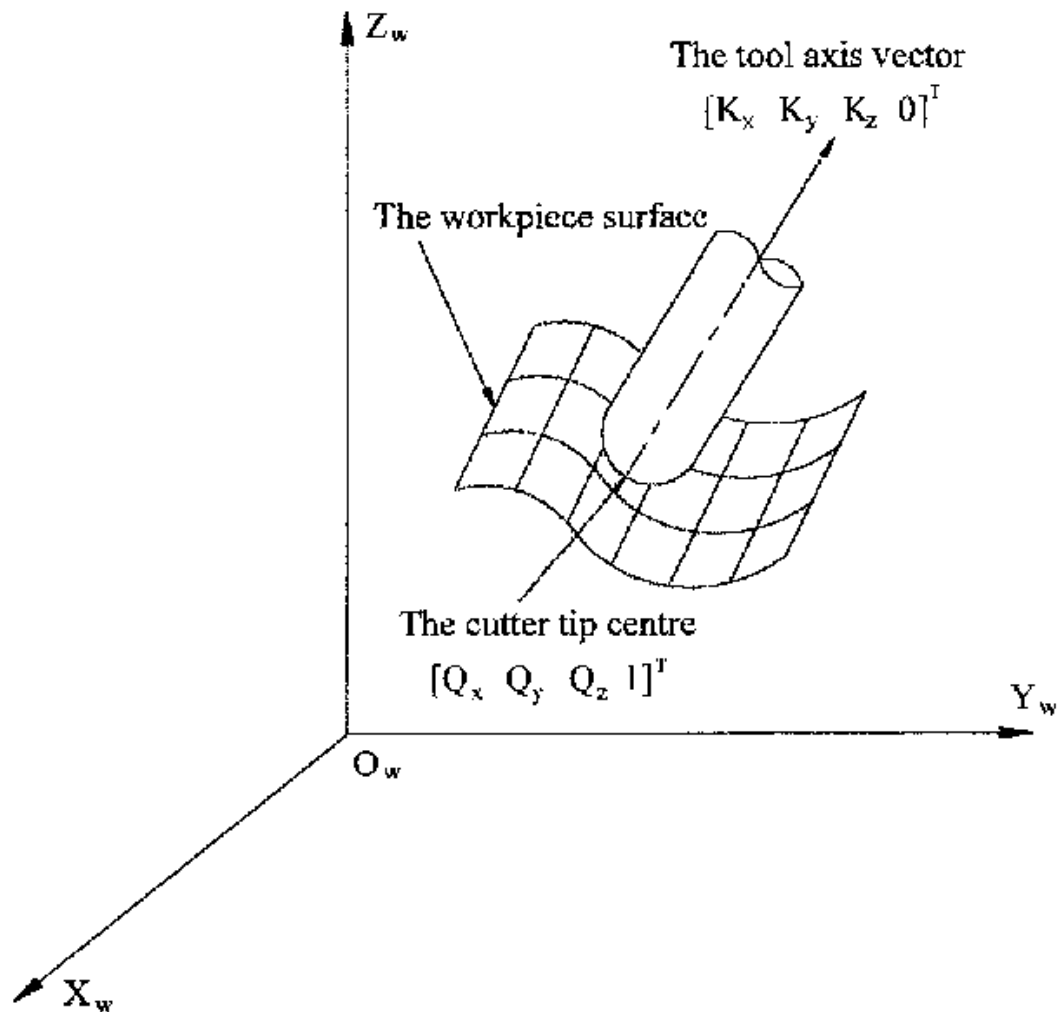


Figure 18.1: Cutter location data

18.4 Translation and Rotation Matrices

Homogeneous transformations provide a simple way to describe the mathematics of multi-axis machine kinematics. A transformation of the space H is a 4×4 matrix and can represent translation and rotation transformations. Given a point x, y, z described by a vector $u = \{x, y, z, 1\}^T$, then its transformation v is represented by the matrix product

$$v = H \cdot u$$

There are four fundamental transformation matrices on which 5-axis kinematics can be based:

$$T(a, b, c) = \begin{bmatrix} 1 & 0 & 0 & a \\ 0 & 1 & 0 & b \\ 0 & 0 & 1 & c \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad R(X, \theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & C\theta & -S\theta & 0 \\ 0 & S\theta & C\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2)$$

$$R(Y, \theta) = \begin{bmatrix} C\theta & 0 & S\theta & 0 \\ 0 & 1 & 0 & 0 \\ -S\theta & 0 & C\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad R(Z, \theta) = \begin{bmatrix} C\theta & -S\theta & 0 & 0 \\ S\theta & C\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3)$$

The matrix $T(a,b,c)$ implies a translation in the X, Y, Z coordinate directions by the amounts a, b, c respectively. The R matrices imply rotations of the angle theta about the X, Y and Z coordinate axes respectively. The C and S symbols refer to cosine and sine functions respectively.

18.5 Table Rotary/Tilting 5-Axis Configurations

In these machine tools the two rotational axes mount on the work table of the machine. Two forms are typically used:

- A rotary table which rotates about the vertical Z-axis (C-rotation, secondary) mounted on a tilting table which rotates about the X- or Y-axis (A- or B-rotation, primary). The workpiece is mounted on the rotary table.
- A tilting table which rotates about the X- or Y-axis (A- or B-rotation, secondary) is mounted on a rotary table which rotates about the Z-axis (C-rotation, primary), with the workpiece on the tilting table.

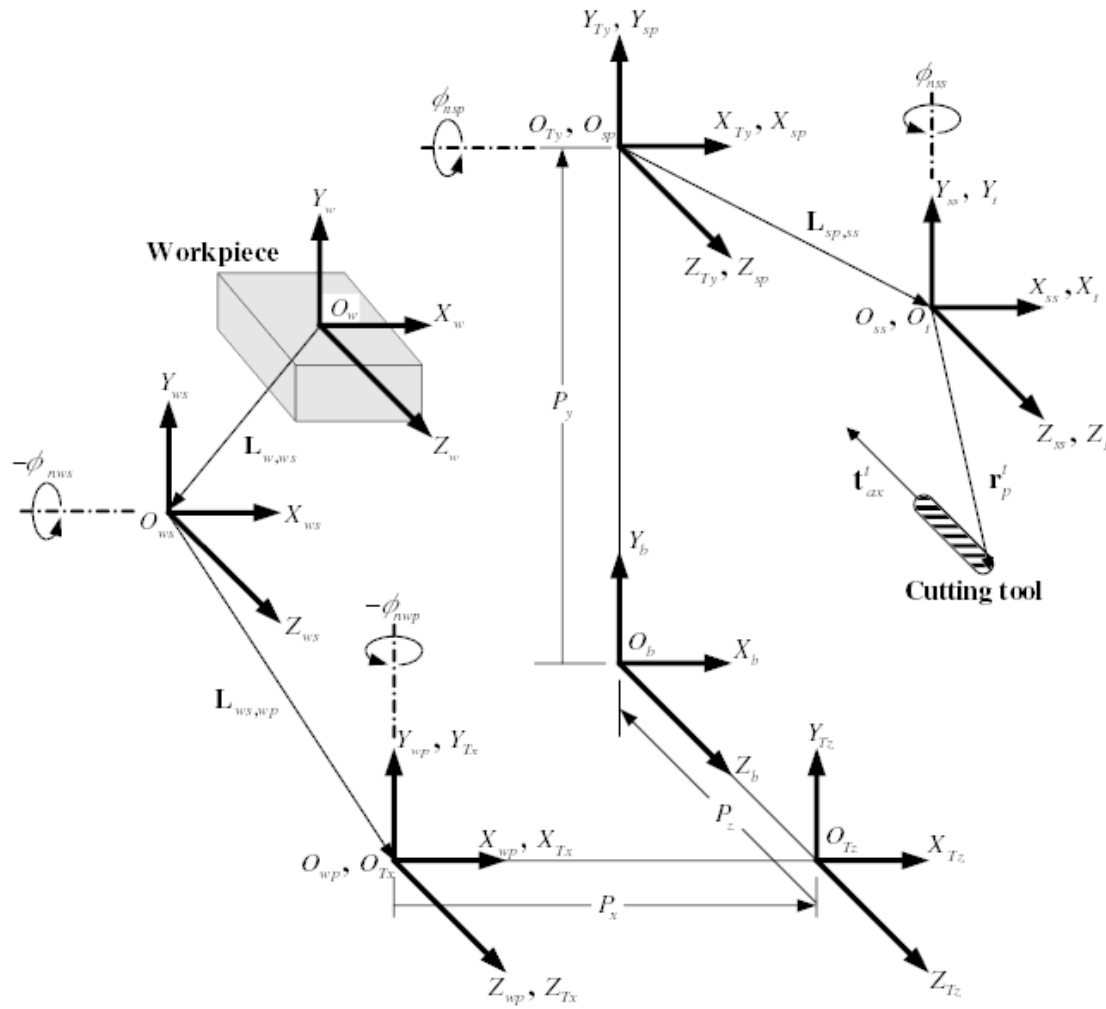


Figure 18.2: General configuration and coordinate systems

A multi-axis machine can be considered to consist of a series of links connected by joints. By embedding a coordinate frame in each link of the machine and using homogeneous transformations, we can describe the relative position and orientation between these coordinate frames

We need to describe a relationship between the workpiece coordinate system and the tool coordinate system. This can be defined by a transformation matrix wA_t , which can be found by subsequent transformations between the different structural elements or links of the machine, each with its own defined coordinate system. In general such a transformation may look as follows:

$${}^wA_t = {}^wA_1 \cdot {}^1A_2 \cdot {}^2A_3 \cdots {}^nA_t \quad (4)$$

where each matrix ${}^{i-1}A_i$ is a translation matrix T or a rotation matrix R of the form (2,3).

Matrix multiplication is a simple process in which the elements of each row of the lefthand matrix A is multiplied by the elements of each column of the righthand matrix B and summed to obtain an element in the result matrix C , ie.

$$C_{ij} = \sum_{k=1}^n A_{ik} B_{kj}; \quad i = 1, n; \quad j = 1, n$$

In Fig. 2 a generic configuration with coordinate systems is shown [4]. It includes table rotary/tilting axes as well as spindle rotary/tilting axes. Only two of the rotary axes are actually used in a machine tool.

First we will develop the transformations for the first type of configuration mentioned above, ie. a table tilting/rotary (trt) type with no rotating axis offsets. We may give it the name xyzac-trt configuration.

We also develop the transformations for the same type (xyzac-trt), but with rotating axis offsets.

Then we develop the transformations for a xyzbc-trt configuration with rotating axis offsets.

18.5.1 Transformations for a xyzac-trt machine tool with work offsets

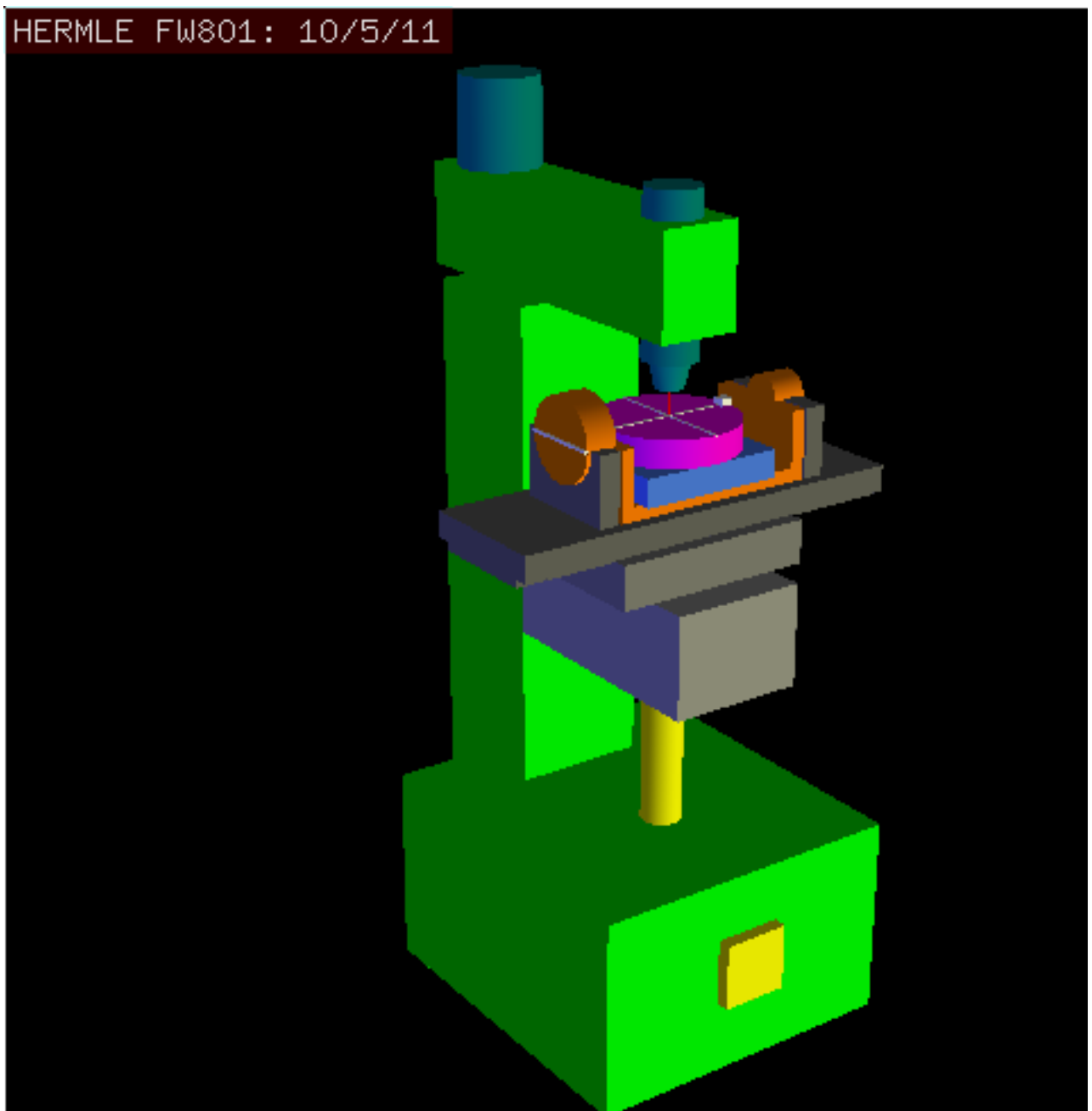


Figure 18.3: vismach model of xyzac-trt with coincident rotation axes

We deal here with a simplified configuration in which the tilting axis and rotary axis intersects at a point called the pivot point as shown in Fig. 4. therefore the two coordinate systems O_{ws} and O_{wp} of Fig. 2 are coincident.

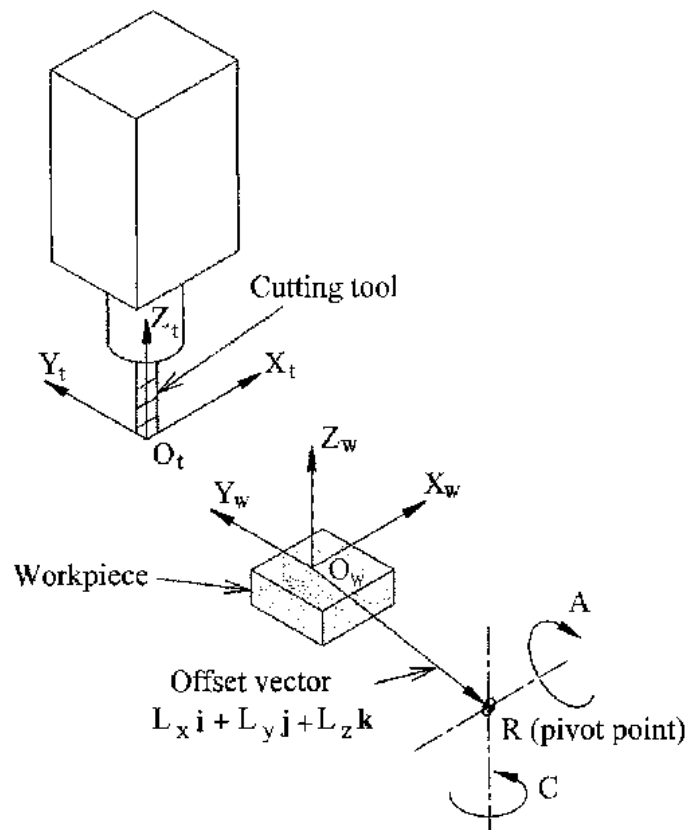


Figure 18.4: Table tilting/rotary configuration

18.5.1.1 Forward transformation

The transformation can be defined by the sequential multiplication of the matrices:

$${}^w A_t = {}^w A_C \cdot {}^C A_A \cdot {}^A A_P \cdot {}^P A_t \quad (5)$$

with the matrices built up as follows:

$${}^w A_C = \begin{bmatrix} 1 & 0 & 0 & L_x \\ 0 & 1 & 0 & L_y \\ 0 & 0 & 1 & L_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad {}^C A_A = \begin{bmatrix} C_C & S_C & 0 & 0 \\ -S_C & C_C & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (6)$$

$${}^A A_P = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & C_A & S_A & 0 \\ 0 & -S_A & C_A & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad {}^P A_t = \begin{bmatrix} 1 & 0 & 0 & P_x \\ 0 & 1 & 0 & P_y \\ 0 & 0 & 1 & P_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (7)$$

In these equations L_x , L_y , L_z defines the offsets of the pivot point of the two rotary axes A and C relative to the workpiece coordinate system origin. Furthermore, P_x , P_y , P_z are the relative distances of the pivot point to the cutter tip position, which can also be called the "joint coordinates" of the pivot point. The pivot point is at the intersection of the two rotary axes. The signs of the S_A and S_C terms are different to those in [2,3] since there the table rotations are negative relative to the workpiece coordinate axes (note that $\sin(-\theta) = -\sin(\theta)$, $\cos(-\theta) = \cos(\theta)$).

When multiplied in accordance with (5), we obtain:

$${}^w A_t = \begin{bmatrix} C_C & S_C C_A & S_C S_A & C_C P_x + S_C C_A P_y + S_C S_A P_z + L_x \\ -S_C & C_C C_A & C_C S_A & -S_C P_x + C_C C_A P_y + C_C S_A P_z + L_y \\ 0 & -S_A & C_A & -S_A P_y + C_A P_z + L_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (8)$$

We can now equate the third column of this matrix with our given tool orientation vector K , ie.:

$$K = \begin{bmatrix} K_x \\ K_y \\ K_z \\ 0 \end{bmatrix} = \begin{bmatrix} S_C S_A \\ C_C S_A \\ C_A \\ 0 \end{bmatrix} \quad (9)$$

From these equations we can solve for the rotation angles θ_A , θ_C . From the third row we find:

$$\theta_A = \cos^{-1}(K_z) \quad (0 < \theta_A < \pi) \quad (10)$$

and by dividing the first row by the second row we find:

$$\theta_C = \tan^{-1}(K_x, K_y) \quad (-\pi < \theta_C < \pi) \quad (11)$$

These relationships are typically used in the CAM post-processor to convert the tool orientation vectors to rotation angles.

Equating the last column of (8) with the tool position vector Q , we can write:

$$Q = \begin{bmatrix} Q_x \\ Q_y \\ Q_z \\ 1 \end{bmatrix} = \begin{bmatrix} C_C P_x + S_C C_A P_y + S_C S_A P_z + L_x \\ -S_C P_x + C_C C_A P_y + C_C S_A P_z + L_y \\ -S_A P_y + C_A P_z + L_z \\ 1 \end{bmatrix} \quad (12)$$

The vector on the right hand side can also be written as the product of a matrix and a vector resulting in:

$$Q = \begin{bmatrix} Q_x \\ Q_y \\ Q_z \\ 1 \end{bmatrix} = \begin{bmatrix} C_C & S_C C_A & S_C S_A & L_x \\ -S_C & C_C C_A & C_C S_A & L_y \\ 0 & -S_A & C_A & L_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} P_x \\ P_y \\ P_z \\ 1 \end{bmatrix} = {}^Q A_P \cdot P \quad (13)$$

This can be expanded to give

$$\begin{aligned} Q_x &= C_c P_x + S_c C_A P_y + S_c S_A P_z + L_x \\ Q_y &= -S_c P_x + C_c C_A P_y + C_c S_A P_z + L_y \\ Q_z &= -S_A P_y + C_A P_z + L_z \end{aligned} \quad (14)$$

which is the *forward transformation* of the kinematics.

18.5.1.2 Inverse Transformation

We can solve for P from equation (13) as $P = ({}^Q A_P)^{-1} * Q$. Noting that the square matrix is a homogeneous 4x4 matrix containing a rotation matrix R and translation vector q, for which the inverse can be written as:

$${}^Q A_P = \begin{bmatrix} R & q \\ 0 & 1 \end{bmatrix} \quad ({}^Q A_P)^{-1} = \begin{bmatrix} R^T & -R^T q \\ 0 & 1 \end{bmatrix} \quad (15)$$

where R^T is the transpose of R (rows and columns swapped). We therefore obtain:

$$\begin{bmatrix} P_x \\ P_y \\ P_z \\ 1 \end{bmatrix} = \begin{bmatrix} C_c & -S_c & 0 & -C_c L_x + S_c L_y \\ S_c C_A & C_c C_A & -S_A & -S_c C_A L_x - C_c C_A L_y + S_A L_z \\ S_c S_A & C_c S_A & C_A & -S_c S_A L_x - C_c S_A L_y - C_A L_z \\ 0 & 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} Q_x \\ Q_y \\ Q_z \\ 1 \end{bmatrix} \quad (16)$$

The desired equations for the *inverse transformation* of the kinematics thus can be written as:

$$\begin{aligned} P_x &= C_c(Q_x - L_x) - S_c(Q_y - L_y) \\ P_y &= S_c C_A(Q_x - L_x) + C_c C_A(Q_y - L_y) - S_A(Q_z - L_z) \\ P_z &= S_c S_A(Q_x - L_x) + C_c S_A(Q_y - L_y) + C_A(Q_z - L_z) \end{aligned} \quad (17)$$

18.5.2 Transformations for a xyzac-trt machine with rotary axis offsets

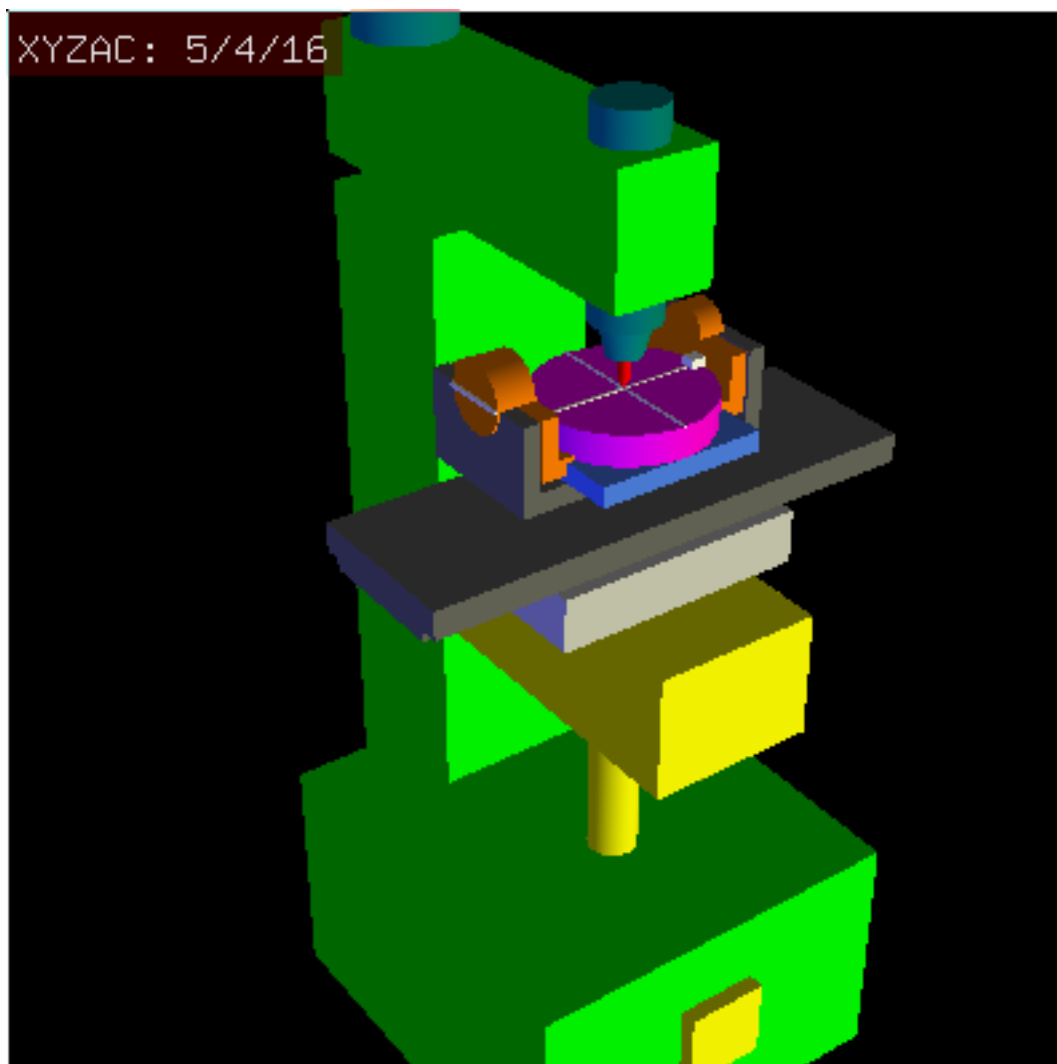


Figure 18.5: vismach model of xyzac-trt with rotational axis offsets (positive)

We deal here with a extended configuration in which the tilting axis and rotary axis do not intersect at a point but have an offset D_y . Furthermore, there is also an z-offset between the two coordinate systems O_{ws} and O_{wp} of Fig. 2, called D_z . A vismach model is shown in Fig. 5 and the offsets are shown in Fig. 6 (positive offsets in this example). To simplify the configuration, the offsets L_x , L_y , L_z of the previous case are not included. They are probably not necessary if one uses the G54 offsets in LinuxCNC by means of the "touch of" facility.

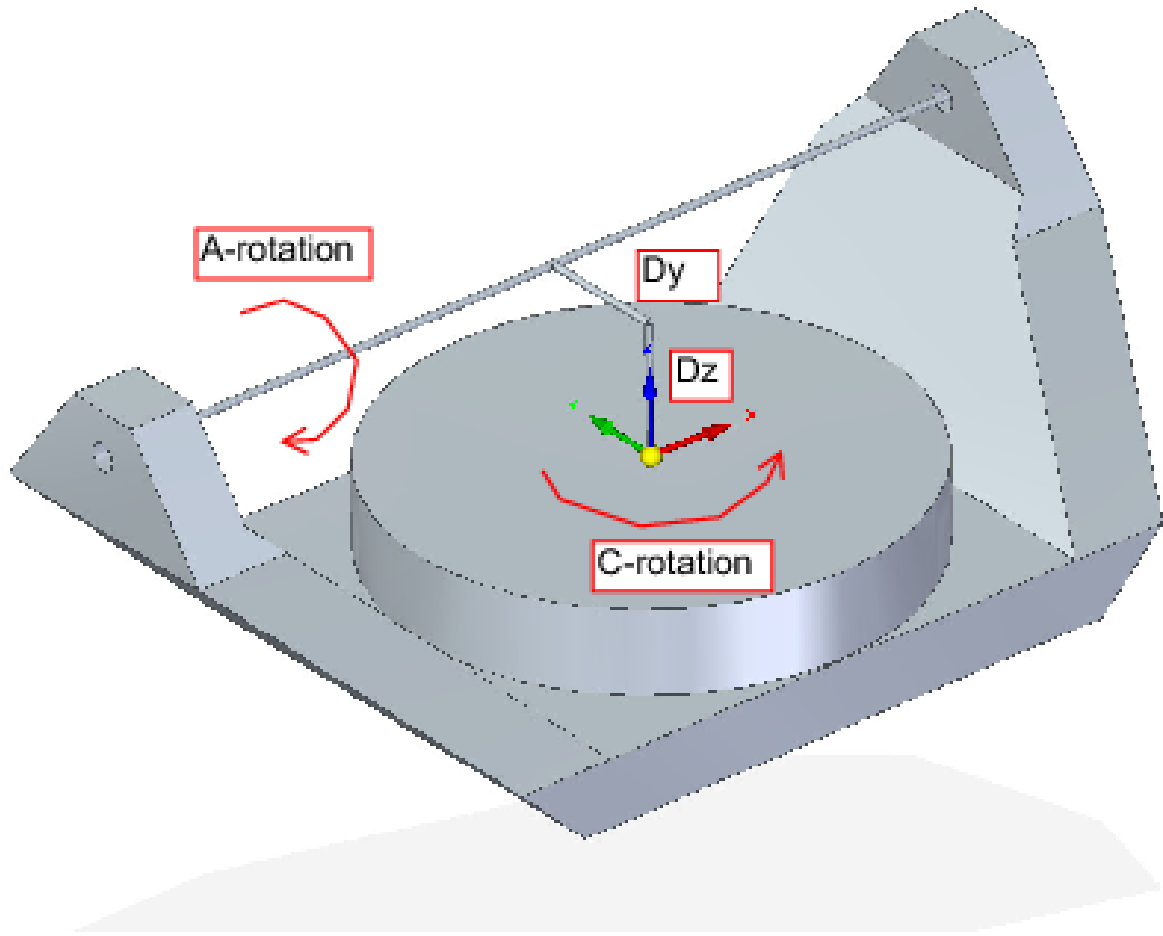


Figure 18.6: Table tilting/rotary xyzac-trt configuration, with axis offsets

18.5.2.1 Forward Transformation

The transformation can be defined by the sequential multiplication of the matrices:

$${}^w A_t = {}^w A_O \cdot {}^O A_A \cdot {}^A A_P \cdot {}^P A_t \quad (18)$$

with the matrices built up as follows:

$${}^w A_O = \begin{bmatrix} C_C & S_C & 0 & 0 \\ -S_C & C_C & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad {}^O A_A = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & D_y \\ 0 & 0 & 1 & D_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (19)$$

$${}^A A_P = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & C_A & S_A & 0 \\ 0 & -S_A & C_A & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad {}^P A_t = \begin{bmatrix} 1 & 0 & 0 & P_x \\ 0 & 1 & 0 & P_y - D_y \\ 0 & 0 & 1 & P_z - D_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (20)$$

In these equations D_y , D_z defines the offsets of the pivot point of the rotary axes A relative to the workpiece coordinate system origin. Furthermore, P_x , P_y , P_z are the relative distances of the pivot point to the cutter tip position, which can also be called the "joint coordinates" of the pivot point. The pivot point is on the A rotary axis.

When multiplied in accordance with (18), we obtain:

$${}^wA_t = \begin{bmatrix} C_C & S_C C_A & S_C S_A & C_C P_x + S_C C_A (P_y - D_y) + S_C S_A (P_z - D_z) + S_C D_y \\ -S_C & C_C C_A & C_C S_A & -S_C P_x + C_C C_A (P_y - D_y) + C_C S_A (P_z - D_z) + C_C D_y \\ 0 & -S_A & C_A & -S_A (P_y - D_y) + C_A (P_z - D_z) + D_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (21)$$

We can now equate the third column of this matrix with our given tool orientation vector K, ie.:

$$K = \begin{bmatrix} K_x \\ K_y \\ K_z \\ 0 \end{bmatrix} = \begin{bmatrix} S_C S_A \\ C_C S_A \\ C_A \\ 0 \end{bmatrix} \quad (22)$$

From these equations we can solve for the rotation angles θ_{A_1} , θ_{A_2} . From the third row we find:

$$\theta_A = \cos^{-1}(K_z) \quad (0 < \theta_A < \pi) \quad (23)$$

and by dividing the second row by the first row we find:

$$\theta_C = \tan^{-1}(K_x, K_y) \quad (-\pi < \theta_C < \pi) \quad (24)$$

These relationships are typically used in the CAM post-processor to convert the tool orientation vectors to rotation angles.

Equating the last column of (21) with the tool position vector Q, we can write:

$$Q = \begin{bmatrix} Q_x \\ Q_y \\ Q_z \\ 1 \end{bmatrix} = \begin{bmatrix} C_C P_x + S_C C_A (P_y - D_y) + S_C S_A (P_z - D_z) + S_C D_y \\ -S_C P_x + C_C C_A (P_y - D_y) + C_C S_A (P_z - D_z) + C_C D_y \\ -S_A (P_y - D_y) + C_A (P_z - D_z) + D_z \\ 1 \end{bmatrix} \quad (25)$$

The vector on the right hand side can also be written as the product of a matrix and a vector resulting in:

$$Q = \begin{bmatrix} Q_x \\ Q_y \\ Q_z \\ 1 \end{bmatrix} = \begin{bmatrix} C_C & S_C C_A & S_C S_A & -S_C C_A D_y - S_C S_A D_z + S_C D_y \\ -S_C & C_C C_A & C_C S_A & -C_C C_A D_y - C_C S_A D_z + C_C D_y \\ 0 & -S_A & C_A & S_A D_y - C_A D_z + D_z \\ 0 & 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} P_x \\ P_y \\ P_z \\ 1 \end{bmatrix} = {}^Q A_P \cdot P \quad (26)$$

which is the *forward transformation* of the kinematics.

18.5.2.2 Inverse Transformation

We can solve for P from equation (25) as $P = ({}^Q A_P)^{-1} \cdot Q$ using (15) as before. We thereby obtain:

$$\begin{bmatrix} P_x \\ P_y \\ P_z \\ 1 \end{bmatrix} = \begin{bmatrix} C_C & -S_C & 0 & 0 \\ S_C C_A & C_C C_A & -S_A & -C_A D_y + S_A D_z + D_y \\ S_C S_A & C_C S_A & C_A & -S_A D_y - C_A D_z + D_z \\ 0 & 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} Q_x \\ Q_y \\ Q_z \\ 1 \end{bmatrix} \quad (27)$$

The desired equations for the *inverse transformation* of the kinematics thus can be written as:

$$\begin{aligned} P_x &= C_C Q_x - S_C Q_y \\ P_y &= S_C C_A Q_x + C_C C_A Q_y - S_A Q_z - C_A D_y + S_A D_z + D_y \\ P_z &= S_C S_A Q_x + C_C S_A Q_y + C_A Q_z - S_A D_y - C_A D_z + D_z \end{aligned} \quad (28)$$

18.5.3 Transformations for a xyzbc-trt machine with rotary axis offsets

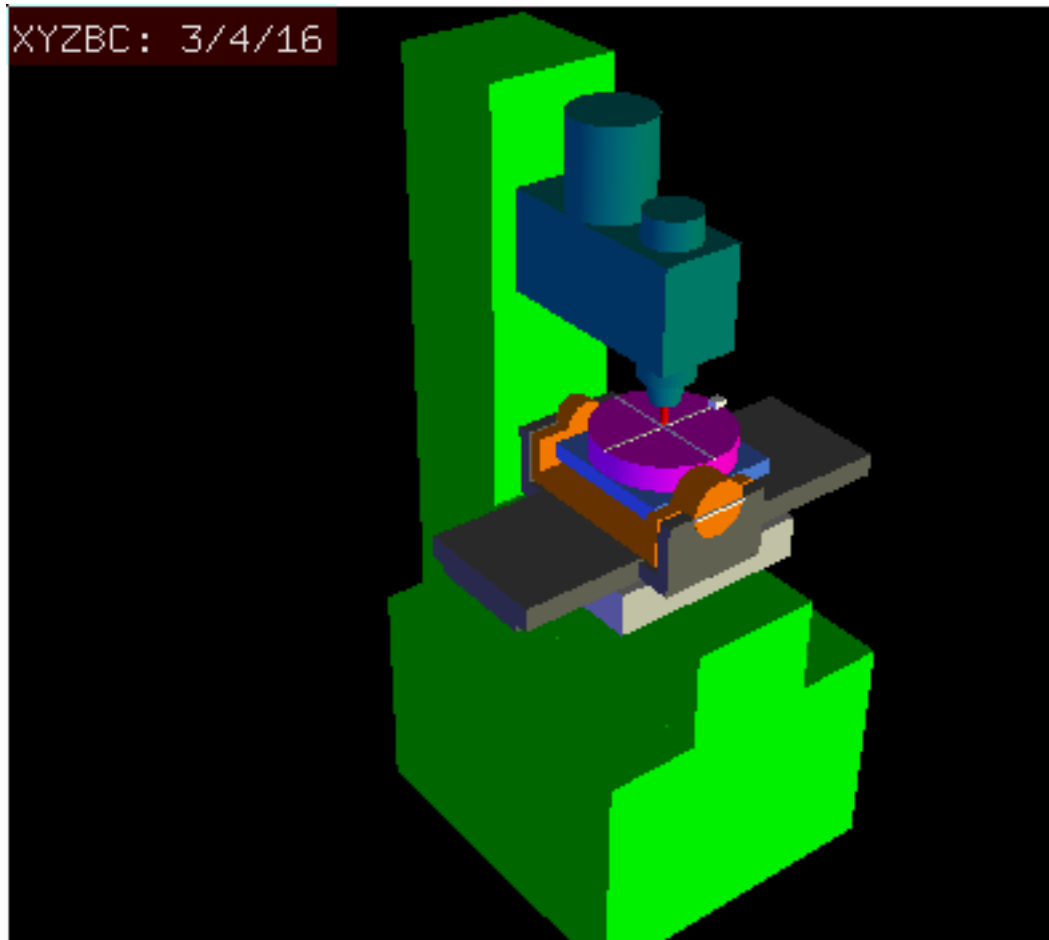


Figure 18.7: vismach model of xyzbc-trt with rotational axis offsets (negative)

We deal here again with a extended configuration in which the tilting axis (about the y-axis) and rotary axis do not intersect at a point but have an offset D_x . Furthermore, there is also an z-offset between the two coordinate systems O_{ws} and O_{wp} of Fig. 2, called D_z . A vismach model is shown in Fig. 7 (negative offsets in this example) and the positive offsets are shown in Fig. 8.

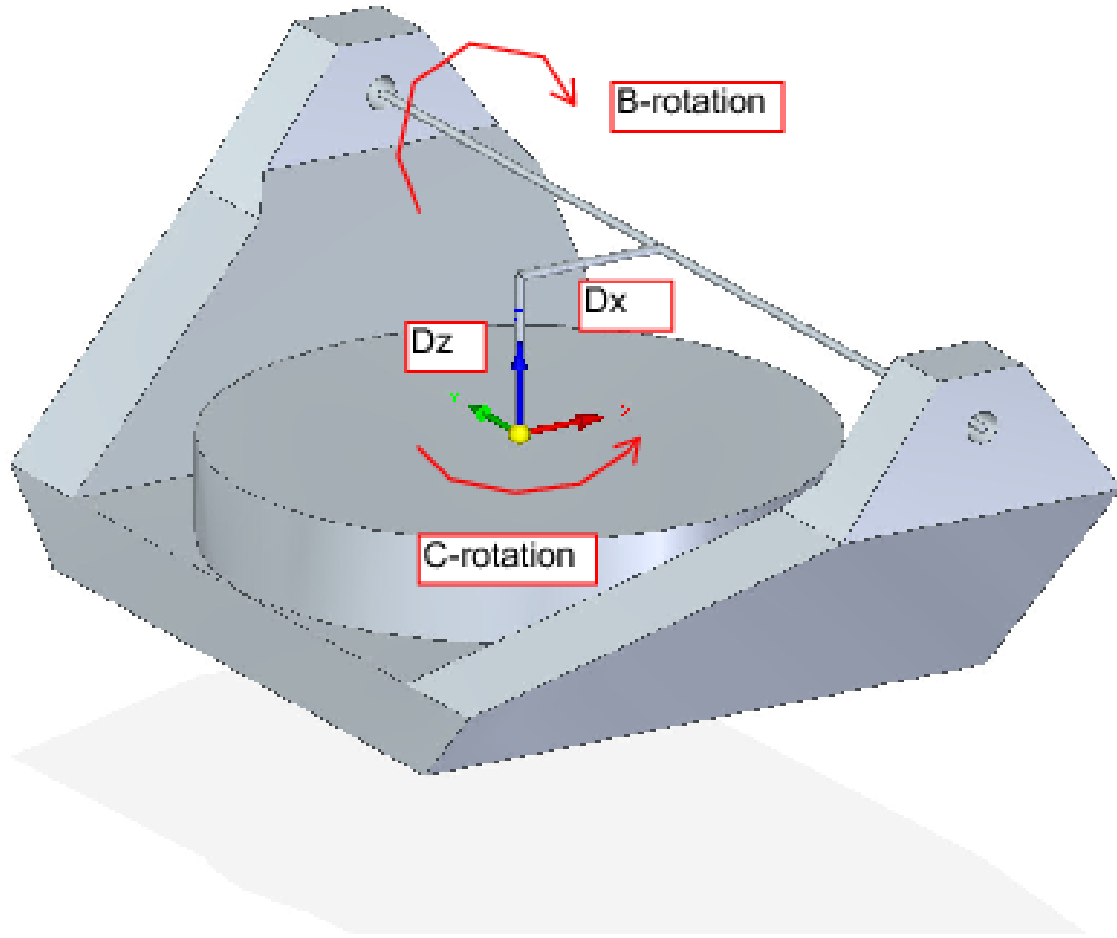


Figure 18.8: Table tilting/rotary xyzbc-trt configuration, with axis offsets

18.5.3.1 Forward Transformation

The transformation can be defined by the sequential multiplication of the matrices:

$${}^w A_t = {}^w A_O \cdot {}^O A_B \cdot {}^B A_P \cdot {}^P A_t \quad (29)$$

with the matrices built up as follows:

$${}^w A_O = \begin{bmatrix} C_C & S_C & 0 & 0 \\ -S_C & C_C & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad {}^O A_B = \begin{bmatrix} 1 & 0 & 0 & D_x \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & D_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (30)$$

$${}^B A_P = \begin{bmatrix} C_B & 0 & -S_B & 0 \\ 0 & 1 & 0 & 0 \\ S_B & 0 & C_B & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad {}^P A_t = \begin{bmatrix} 1 & 0 & 0 & P_x - D_x \\ 0 & 1 & 0 & P_y \\ 0 & 0 & 1 & P_z - D_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (31)$$

In these equations D_x , D_z defines the offsets of the pivot point of the rotary axes B relative to the workpiece coordinate system origin. Furthermore, P_x , P_y , P_z are the relative distances of the pivot point to the cutter tip position, which can also be called the "joint coordinates" of the pivot point. The pivot point is on the B rotary axis.

When multiplied in accordance with (29), we obtain:

$${}^w A_t = \begin{bmatrix} C_C C_B & S_C & -C_C S_B & C_C C_B (P_x - D_x) + S_C P_y - C_C S_B (P_z - D_z) + C_C D_x \\ -S_C C_B & C_C & S_C S_B & -S_C C_B (P_x - D_x) + C_C P_y + S_C S_B (P_z - D_z) - S_C D_x \\ S_B & 0 & C_B & S_B (P_x - D_x) + C_B (P_z - D_z) + D_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (32)$$

We can now equate the third column of this matrix with our given tool orientation vector K , i.e.:

$$K = \begin{bmatrix} K_x \\ K_y \\ K_z \\ 0 \end{bmatrix} = \begin{bmatrix} -C_C S_B \\ S_C S_B \\ C_B \\ 0 \end{bmatrix} \quad (33)$$

From these equations we can solve for the rotation angles θ_B , θ_C . From the third row we find:

$$\theta_B = \cos^{-1}(K_z) \quad (0 < \theta_B < \pi) \quad (34)$$

and by dividing the second row by the first row we find:

$$\theta_C = \tan^{-1}(K_y, K_x) \quad (-\pi < \theta_C < \pi) \quad (35)$$

These relationships are typically used in the CAM post-processor to convert the tool orientation vectors to rotation angles.

Equating the last column of (32) with the tool position vector Q , we can write:

$$Q = \begin{bmatrix} Q_x \\ Q_y \\ Q_z \\ 1 \end{bmatrix} = \begin{bmatrix} C_C C_B (P_x - D_x) + S_C P_y - C_C S_B (P_z - D_z) + C_C D_x \\ -S_C C_B (P_x - D_x) + C_C P_y + S_C S_B (P_z - D_z) - S_C D_x \\ S_B (P_x - D_x) + C_B (P_z - D_z) + D_z \\ 1 \end{bmatrix} \quad (36)$$

The vector on the right hand side can also be written as the product of a matrix and a vector resulting in:

$$Q = \begin{bmatrix} Q_x \\ Q_y \\ Q_z \\ 1 \end{bmatrix} = \begin{bmatrix} C_C C_B & S_C & -C_C S_B & -C_C C_B D_x + C_C S_B D_z + C_C D_x \\ -S_C C_B & C_C & S_C S_B & S_C C_B D_x - S_C S_B D_z - S_C D_x \\ S_B & 0 & C_B & -S_B D_x - C_B D_z + D_z \\ 0 & 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} P_x \\ P_y \\ P_z \\ 1 \end{bmatrix} = {}^Q A_P \cdot P \quad (37)$$

which is the *forward transformation* of the kinematics.

18.5.3.2 Inverse Transformation

We can solve for P from equation (37) as $P = ({}^Q A_P)^{-1} * Q$.

With the same approach as before, we obtain:

$$\begin{bmatrix} P_x \\ P_y \\ P_z \\ 1 \end{bmatrix} = \begin{bmatrix} C_C C_B & -S_C C_B & S_B & -C_B D_x - S_B D_z + D_x \\ S_C & C_C & 0 & 0 \\ -C_C S_B & S_C S_B & C_B & S_B D_x - C_B D_z + D_z \\ 0 & 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} Q_x \\ Q_y \\ Q_z \\ 1 \end{bmatrix} \quad (38)$$

The desired equations for the *inverse transformation* of the kinematics thus can be written as:

$$\begin{aligned} P_x &= C_C C_B Q_x - S_C C_B Q_y + S_B Q_z - C_B D_x - S_B D_z + D_x \\ P_y &= S_C Q_x + C_C Q_y \\ P_z &= -C_C S_B Q_x + S_C S_B Q_y + C_B Q_z + S_B D_x - C_B D_z + D_z \end{aligned} \quad (39)$$

18.6 Table Rotary/Tilting Examples

LinuxCNC includes kinematics modules for the *xyzac-trt* and *xyzbc-trt* topologies described in the mathematics detailed above. For interested users, the source code is available in the git tree in the *src/emc/kinematics/* directory.

Example *xyzac-trt* and *xyzbc-trt* simulation configurations are located in the Sample Configurations (*configs/sim/axis/vismach/5axis/table-rotary-tilting/*) directory.

The example configurations include the required INI files and an examples subdirectory with gode (.ngc) files. These sim configurations invoke a realistic 3-dimensional model using the LinuxCNC vismach facility.

18.6.1 Vismach Simulation Models

Vismach is a library of python routines to display a dynamic simulation of a CNC machine on the PC screen. The python script for a particular machine is loaded in HAL and data passed by HAL pin connections. The user-space vismach model is loaded by a HAL command like:

```
loadusr -W xyzac-trt-gui
```

and connections are made using HAL commands like:

```
net :table-x joint.0.pos-fb xyzac-trt-gui.table-x
net :saddle-y joint.1.pos-fb xyzac-trt-gui.saddle-y
...
```

See the simulation INI files for details of the HAL connections used for the vismach model.

18.6.2 Tool-Length Compensation

In order to use tools from a tool table sequentially with tool-length compensation applied automatically, a further Z-offset is required. For a tool that is longer than the "master" tool, which typically has a tool length of zero, LinuxCNC has a variable called "motion.tooloffset.z". If this variable is passed on to the kinematic component (and vismach python script), then the necessary additional Z-offset for a new tool can be accounted for by adding the component statement, for example:

$$D_z = D_z + \text{tool-offset}$$

The required HAL connection (for *xyzac-trt*) is:

```
net :tool-offset motion.tooloffset.z xyzac-trt-kins.tool-offset
```

where:

```
:tool-offset ----- signal name
motion.tooloffset.z ----- output HAL pin from LinuxCNC motion module
xyzac-trt-kins.tool-offset -- input HAL pin to xyzac-trt-kins
```

18.7 Custom Kinematics Components

LinuxCNC implements kinematics using a HAL component that is loaded at startup of LinuxCNC. The most common kinematics module, *trivkins*, implements identity (trivial) kinematics where there is a one-to-one correspondence between an axis coordinate letter and a motor joint. Additional kinematics modules for more complex systems (including *xyzac-trt* and *xyzbc-trt* described above) are available.

See the kins manpage (\\$ **man kins**) for brief descriptions of the available kinematics modules.

The kinematics modules provided by LinuxCNC are typically written in the C-language. Since a standard structure is used, creation of a custom kinematics module is facilitated by copying an existing source file to a user file with a new name, modifying it, and then installing.

Installation is done using `halcompile`:

```
sudo halcompile --install kinsname.c
```

where “kinsname” is the name you give to your component. The `sudo` prefix is required to install it and you will be asked for your root password. See the `halcompile` man page for more information (\\$ **man halcompile**)

Once it is compiled and installed you can reference it in your config setup of your machine. This is done in the INI file of your config directory. For example, the common INI specification:

```
[KINS]
KINEMATICS = trivkins
```

is replaced by

```
[KINS]
KINEMATICS = kinsname
```

where “kinsname” is the name of your kins program. Additional HAL pins may be created by the module for variable configuration items such as the D_x , D_y , D_z , tool-offset used in the *xyzac-trt* kinematics module. These pins can be connected to a signal for dynamic control or set once with HAL connections like:

```
# set offset parameters
net :tool-offset motion.tooloffset.z xyzac-trt-kins.tool-offset
setp xyzac-trt-kins.y-offset 0
setp xyzac-trt-kins.z-offset 20
```


18.8 Figures

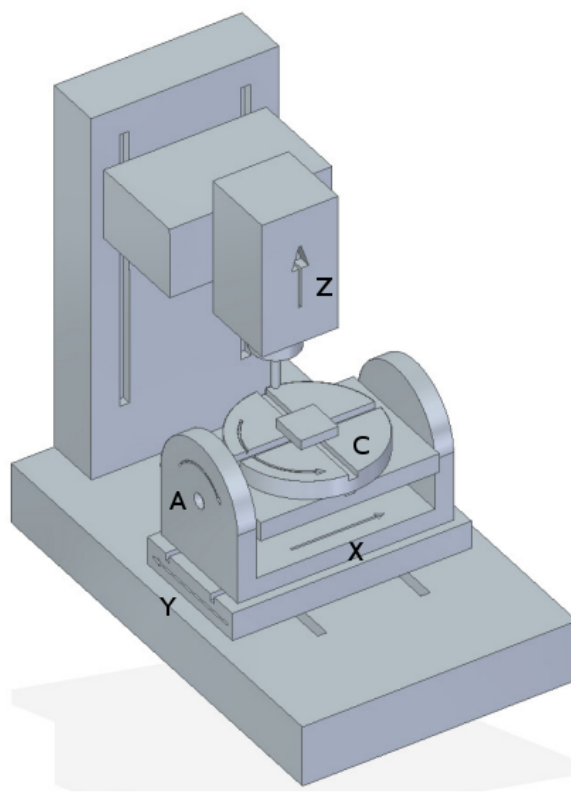


Figure 18.9: Table tilting/rotating configuration

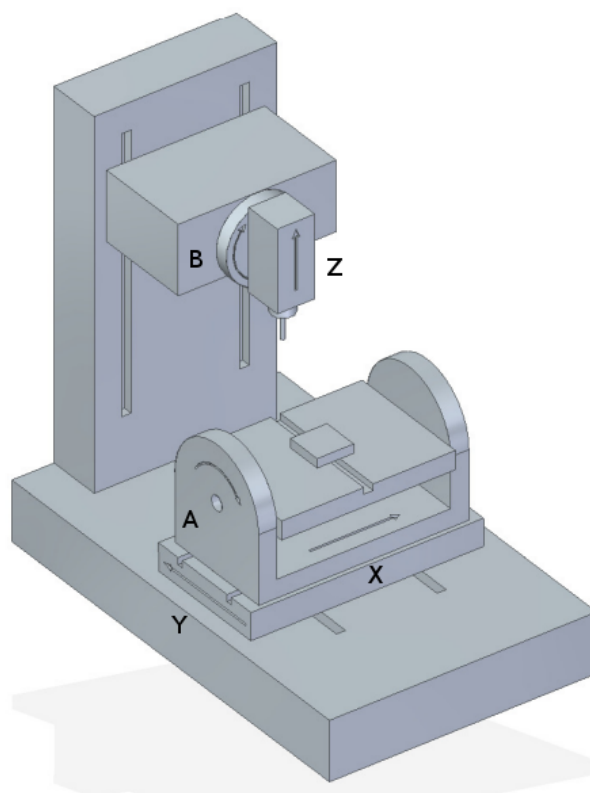


Figure 18.10: Spindle/table tilting configuration



Figure 18.11: Spindle tilting/rotary configuration

18.9 REFERENCES

1. A Postprocessor Based on the Kinematics Model for General Five-Axis machine Tools: C-H She, R-S Lee, J Manufacturing Processes, V2 N2, 2000.
2. NC Post-processor for 5-axis milling of table-rotating/tilting type: YH Jung, DW Lee, JS Kim, HS Mok, J Materials Processing Technology, 130-131 (2002) 641-646.
3. 3D 6-DOF Serial Arm Robot Kinematics, RJ du Preez, SA-CNC-CLUB, Dec. 5, 2013.
4. Design of a generic five-axis postprocessor based on generalized kinematics model of machine tool: C-H She, C-C Chang, Int. J Machine Tools & Manufacture, 47 (2007) 537-545.

Chapter 19

Switchable Kinematics (switchkins)

19.1 Introduction

A number of kinematics modules support the switching of kinematics calculations. These modules support a default kinematics method (type0), a second built-in method (type1), and (optionally) a user-provided kinematics method (type2). Identity kinematics are typically used for the type1 method.

The switchkins functionality can be used for machines where post-homing joint control is needed during setup or to avoid movement near singularities from G-code. Such machines use specific kinematics calculations for most operations but can be switched to identity kinematics for control of individual joints after homing.

The kinematics type is selected by a motion module HAL pin that can be updated from a G-code program or by interactive MDI commands. The halui provisions for activating MDI commands can be used to allow buttons to select the kinematics type using hardware controls or a virtual panel (PyVCP, GladeVCP, etc.).

When a kinematics type is changed, the G-code must also issue commands to **force synchronization** of the interpreter and motion parts of LinuxCNC. Typically, a HAL pin *read* command (M66 E0 L0) is used immediately after altering the controlling HAL pin to force synchronization.

19.2 Switchable Kinematic Modules

The following kinematics modules support switchable kinematics:

1. **xyzac-trt-kins** (type0:xyzac-trt-kins type1:identity)
2. **xyzbc-trt-kins** (type0:xyzbc-trt-kins type1:identity)
3. **genhexkins** (type0:genhexkins type1:identity)
4. **genserkins** (type0:genserkins type1:identity) (puma560 example)
5. **pumakins** (type0:pumakins type1:identity)
6. **scarakins** (type0:scarakins type1:identity)
7. **5axiskins** (type0:5axiskins type1:identity) (bridgemill)

The xyz[ab]c-trt-kins modules by default use type0==xyz[ab]c-trt-kins for backwards compatibility. The provided sim configs alter the type0/type1 convention by forcing type0==identity kinematics using the module string parameter *sparm* with an INI file setting like:

```
[KINS]
KINEMATICS = xyzac-trt-kins sparm=identityfirst
...
```

19.2.1 Identity letter assignments

When using an **identity** kinematics type, the module parameter *coordinates* can be used to assign letters to joints in arbitrary order from the set of allowed coordinate letters. Examples:

```
[KINS]
JOINTS = 6

# conventional identity ordering: joint0==x, joint1==y, ...
KINEMATICS = genhexkins coordinates=xyzabc

# custom identity ordering: joint0==c, joint1==b, ...
# KINEMATICS = genhexkins coordinates=cbazyx
```

Note

If the *coordinates=* parameter is omitted, the default joint-letter identity assignments are *joint0==x,joint1==y,...*

The joint assignments provided for **identity** kinematics when using the *coordinates* parameter are identical to those provided for the *trivkins* module. However, duplication of axis letters to assign multiple joints for a coordinate letter is not generally applicable for serial or parallel kinematics (like *genserkins*, *pumakins*, *genhexkins*, etc.) where there is no simple relationship between joints and coordinates.

Duplication of axis coordinate letters is supported in the kinematics modules *xyzac-trt-kins*, *xyzbc-trt-kins*, and *5axiskins* (*bridgemill*). Typical applications for duplicate coordinates are gantry machines where two motors (joints) are used for the transverse axis.

19.2.2 Backwards compatibility

Switchable kinematics initialize with *motion.switchkins-type==0* implementing their eponymous kinematics method. If the *motion.switchkins-type* pin is not connected — as in legacy configurations — only the default kinematics type is available.

19.3 HAL Pins

Kinematics switching is controlled by the motion module input HAL pin **motion.switchkins-type**. The floating point pin value is truncated to integer and used to select one of the provided kinematics types. The zero startup value selects the *type0* default kinematics type.

Note

The *motion.switchkins-type* input pin is floating point in order to facilitate connections to motion module output pins like *motion.analog-out-0n* that are controllable by standard M-codes (typically M68EnL0).

Output HAL pins are provided to inform GUIs of the current kinematics type. These pins can also be connected to digital inputs that are read by G-code programs to enable or disable program behavior in accordance with the active kinematics type.

19.3.1 HAL Pin Summary

1. **motion.switchkins-type** Input (float)
2. **kinstype.is-0** Output (bit)
3. **kinstype.is-1** Output (bit)
4. **kinstype.is-2** Output (bit)

19.4 Usage

19.4.1 HAL Connections

Switchkins functionality is enabled by the pin **motion.switchkins-type**. Typically, this pin is sourced by an analog output pin like `motion.analog-out-03` so that it can be set by M68 commands. Example:

```
net :kinstype-select <= motion.analog-out-03
net :kinstype-select => motion.switchkins-type
```

19.4.2 G-/M-code commands

Kinstype selection is managed using G-code sequences like:

```
...
M68 E3 Q1 ;update analog-out-03 to select kinstype 1
M66 E0 L0 ;sync interp-motion
...
...      ;user G-code
...
M68 E3 Q0 ;update analog-out-03 to select kinstype 0
M66 E0 L0 ;sync interp-motion
...
```

Note

An M66 *wait-on-input* command updates the #5399 variable. If the current value of this variable is needed for subsequent purposes, it should be copied to an additional variable before invoking M66.

These G-code command sequences are typically implemented in G-code subroutines as remapped M-codes or with conventional M-code scripts.

Suggested codes (as used in sim configs) are:

Conventional User M-codes:

1. M128 Select kinstype 0 (startup default kinematics)
2. M129 Select kinstype 1 (typically identity kinematics)
3. M130 Select kinstype 2 (user-provided kinematics)

Remapped M-codes:

1. M428 Select kintype 0 (startup default kinematics)
2. M429 Select kintype 1 (typically identity kinematics)
3. M430 Select kintype 2 (user-provided kinematics)

Note

Conventional user M-codes (in the range M100-M199) are in modal group 10. Remapped M-codes (in the range M200 to M999) can specify a modalgroup. See the remap documentation for additional information.

19.4.3 INI file limit settings

LinuxCNC trajectory planning uses limits for position (min,max), velocity, and acceleration for each applicable coordinate letter specified in the configuration INI file. Example for letter L (in the set XYZABCUVW):

```
[AXIS_L]
MIN_LIMIT =
MAX_LIMIT =
MAX_VELOCITY =
MIN_ACCELERATION =
```

The INI file limits specified apply to the type 0 default kinematics type that is activated at startup. These limits may **not** be applicable when switching to alternative kinematics. However, since an interpreter-motion synchronization is required when switching kinematics, INI-HAL pins can be used to setup limits for a pending kinematics type.

Note

INI-HAL pins are typically not recognized during a G-code program unless a synchronization (queue-buster) command is issued. See the milltask manpage for more information (\$ man milltask)

The relevant INI-HAL pins for a joint number (N) are:

```
ini.N.min_limit
ini.N.max_limit
ini.N.max_acceleration
ini.N.max_velocity
```

The relevant ini-HAL pins for an axis coordinate (L) are:

```
ini.L.min_limit
ini.L.max_limit
ini.L.max_velocity
ini.L.max_acceleration
```

Note

In general, there are no fixed mappings between joint numbers and axis coordinate letters. There may be specific mappings for some kinematics modules especially those that implement identity kinematics (trivkins). See the kins man page for more information (\$ man kins)

A user-provided M-code can alter any or all of the axis coordinate limits prior to changing the motion.switchkin type pin and synchronizing the interpreter and motionparts of LinuxCNC. As an example, a bash script invoking halcmd can be *hardcoded* to set any number of HAL pins:

```
#!/bin/bash
halcmd -f <<EOF
setp ini.x.min_limit -100
setp ini.x.max_limit 100
# ... repeat for other limit parameters
EOF
```

Scripts like this can be invoked as a user M-code and used **prior** to the kins switching M-code that updates the motion.switchkins-type HAL pin and forces an interp-motion sync. Typically, separate scripts would be used for each kintype (0,1,2).

When identity kinematics are provided as a means to control individual joints, it may be convenient to set or restore limits as specified in the system INI file. For example, a robot starts with a complex (non-identity) kinematics (type0) after homing. The system is configured so that it can be switched to identity kinematics (type1) in order to manipulate individual joints using the conventional letters from the set *XYZABCUVW*. The INI file settings ([*AXIS_L*]) are **not** applicable when operating with identity (type1) kinematics. To address this use case, the user M-code scripts can be designed as follows:

M129 (Switch to identity type1)

1. read and parse INI file
2. HAL: setp the INI-HAL limit pins for each axis letter ([*AXIS_L*]) according to the *identity-referenced* joint number INI file setting ([*JOINT_N*])
3. HAL: setp motion.switchkins-type 1
4. MDI: execute a syncing G-code (M66E0L0)

M128 (restore robot default kinematics type 0)

1. read and parse INI file
2. HAL: setp the INI-HAL limit pins for each axis letter ([*AXIS_L*]) according to the appropriate INI file setting ([*AXIS_L*])
3. HAL: setp motion.switchkins-type 0
4. MDI: execute a syncing G-code (M66E0L0)

Note

The vismach simulation configurations for a puma robot demonstrate M-code scripts (M128,M129,M130) for this example use case.

19.4.4 Offset considerations

Like INI file limit settings, coordinate system offsets (G92, G10L2, G10L20, G43, etc) are generally applicable only for the type 0 default startup kinematics type. When switching kinematics types, it may be **important** to either reset all offsets prior to switching or update offsets based on system-specific requirements.

19.5 Simulation configs

Simulation configs (requiring no hardware) are provided with illustrative vismach displays in subdirectories of configs/sim/axis/vismach/ .

1. 5axis/table-rotary-tilting/xyzac-trt.ini (xyzac-trt-kins)
2. 5axis/table-rotary-tilting/xyzbc-trt.ini (xyzac-trt-kins)
3. 5axis/bridgemill/5axis.ini (5axiskins)
4. scara/scara.ini (scarakins)
5. puma/puma560.ini (genserkins)
6. puma/puma.ini (pumakins)
7. hexapod-sim/hexapod.ini (genhexkins)

19.6 User kinematics provisions

Custom kinematics can be coded and tested on Run-In-Place (*RIP*) builds. A template file src/emc/kinematics/userkfuncs.c is provided in the distribution. This file can be copied/renamed to a user directory and edited to supply custom kinematics with kintype==2.

The user custom kinematics file can be compiled from out-of-tree source locations for rt-preempt implementations or by replacing the in-tree template file (src/emc/kinematics/userkfuncs.c) for rtai systems.

Preempt-rt make example:

```
$ userkfuncs=/home/myname/kins/mykins.c make && sudo make setuid
```

19.7 Warnings

Unexpected behavior can result if a G-code program is inadvertently started with an incompatible kinematics type. Unwanted behavior can be circumvented in G-code programs by:

1. Connecting appropriate kintype.is.N HAL pins to digital input pins (like motion.digital-in-0m).
2. Reading the digital input pin (M66 E0 Pm) at the start of the G-code program
3. Aborting (M2) the G-code program with a message (DEBUG, problem_message) if the kintype is not suitable.

When using jogging facilities or MDI commands interactively, operator caution is required. Guis should include indicators to display the current kinematics type.

Note

Switching kinematics can cause substantial operational changes requiring careful design, testing, and training for deployment. The management of coordinate offsets, tool compensation, and INI file limits may require complicated and non-standard operating protocols.

19.8 Code Notes

Kinematic modules providing switchkins functionality are linked to the switchkins.o object (switchkins.c) that provides the module *main* program (rtapi_app_main()) and related functions. This *main* program reads (optional) module command-line parameters (coordinates, sparm) and passes them to the module-provided function switchkinsSetup().

The switchkinsSetup() function identifies kintype-specific setup routines and the functions for forward and inverse calculation for each kintype (0,1,2) and sets a number of configuration settings.

After calling switchkinsSetup(), rtapi_app_main() checks the supplied parameters, creates a HAL component, and then invokes the setup routine identified for each kintype (0,1,2).

Each kintype (0,1,2) setup routine can (optionally) create HAL pins and set them to default values. When all setup routines finish, rtapi_app_main() issues hal_ready() for the component to complete creation of the module.

Chapter 20

PID Tuning

20.1 PID Controller

A proportional-integral-derivative controller (PID controller) is a common feedback loop component in industrial control systems.¹

The Controller compares a measured value from a process (typically an industrial process) with a reference set point value. The difference (or *error* signal) is then used to calculate a new value for a manipulable input to the process that brings the process measured value back to its desired set point.

Unlike simpler control algorithms, the PID controller can adjust process outputs based on the history and rate of change of the error signal, which gives more accurate and stable control. (It can be shown mathematically that a PID loop will produce accurate, stable control in cases where a simple proportional control would either have a steady-state error or would cause the process to oscillate).

20.1.1 Control loop basics

Intuitively, the PID loop tries to automate what an intelligent operator with a gauge and a control knob would do. The operator would read a gauge showing the output measurement of a process, and use the knob to adjust the input of the process (the *action*) until the process's output measurement stabilizes at the desired value on the gauge.

In older control literature this adjustment process is called a *reset* action. The position of the needle on the gauge is a *measurement*, *process value* or *process variable*. The desired value on the gauge is called a *set point* (also called *set value*). The difference between the gauge's needle and the set point is the *error*.

A control loop consists of three parts:

1. Measurement by a sensor connected to the process (e.g. encoder),
2. Decision in a controller element,
3. Action through an output device such as an motor.

As the controller reads a sensor, it subtracts this measurement from the *set point* to determine the *error*. It then uses the error to calculate a correction to the process's input variable (the *action*) so that this correction will remove the error from the process's output measurement.

¹This Subsection is taken from an much more extensive article found at http://en.wikipedia.org/wiki/PID_controller

In a PID loop, correction is calculated from the error in three ways: cancel out the current error directly (Proportional), the amount of time the error has continued uncorrected (Integral), and anticipate the future error from the rate of change of the error over time (Derivative).

A PID controller can be used to control any measurable variable which can be affected by manipulating some other process variable. For example, it can be used to control temperature, pressure, flow rate, chemical composition, speed, or other variables. Automobile cruise control is an example of a process outside of industry which utilizes crude PID control.

Some control systems arrange PID controllers in cascades or networks. That is, a *master* control produces signals used by *slave* controllers. One common situation is motor controls: one often wants the motor to have a controlled speed, with the *slave* controller (often built into a variable frequency drive) directly managing the speed based on a proportional input. This *slave* input is fed by the *master* controller's output, which is controlling based upon a related variable.

20.1.2 Theory

PID is named after its three correcting calculations, which all add to and adjust the controlled quantity. These additions are actually *subtractions* of error, because the proportions are usually negative:

Proportional To handle the present, the error is multiplied by a (negative) constant *P* (for *proportional*), and added to (subtracting error from) the controlled quantity. *P* is only valid in the band over which a controller's output is proportional to the error of the system. Note that when the error is zero, a proportional controller's output is zero.

Integral To learn from the past, the error is integrated (added up) over a period of time, and then multiplied by a (negative) constant *I* (making an average), and added to (subtracting error from) the controlled quantity. *I* averages the measured error to find the process output's average error from the set point. A simple proportional system either oscillates, moving back and forth around the set point because there's nothing to remove the error when it overshoots, or oscillates and/or stabilizes at a too low or too high value. By adding a negative proportion of (i.e. subtracting part of) the average error from the process input, the average difference between the process output and the set point is always being reduced. Therefore, eventually, a well-tuned PID loop's process output will settle down at the set point.

Derivative To handle the future, the first derivative (the slope of the error) over time is calculated, and multiplied by another (negative) constant *D*, and also added to (subtracting error from) the controlled quantity. The derivative term controls the response to a change in the system. The larger the derivative term, the more rapidly the controller responds to changes in the process's output.

More technically, a PID loop can be characterized as a filter applied to a complex frequency-domain system. This is useful in order to calculate whether it will actually reach a stable value. If the values are chosen incorrectly, the controlled process input can oscillate, and the process output may never stay at the set point.

20.1.3 Loop Tuning

Tuning a control loop is the adjustment of its control parameters (gain/proportional band, integral gain/reset, derivative gain/rate) to the optimum values for the desired control response. The optimum behavior on a process change or set point change varies depending on the application. Some processes must not allow an overshoot of the process variable from the set point. Other processes must minimize the energy expended in reaching a new set point. Generally stability of response is required and the process must not oscillate for any combination of process conditions and set points.

Tuning of loops is made more complicated by the response time of the process; it may take minutes or hours for a set point change to produce a stable effect. Some processes have a degree of non-linearity and so parameters that work well at full-load conditions don't work when the process is starting up from no-load. This section describes some traditional manual methods for loop tuning.

There are several methods for tuning a PID loop. The choice of method will depend largely on whether or not the loop can be taken *offline* for tuning, and the response speed of the system. If the system can be taken offline, the best tuning method often involves subjecting the system to a step change in input, measuring the output as a function of time, and using this response to determine the control parameters.

Simple method If the system must remain on line, one tuning method is to first set the I and D values to zero. Increase the P until the output of the loop oscillates. Then increase I until oscillation stops. Finally, increase D until the loop is acceptably quick to reach its reference. A fast PID loop tuning usually overshoots slightly to reach the set point more quickly; however, some systems cannot accept overshoot.

Parameter	Rise Time	Overshoot	Settling Time	Steady State Error
P	Decrease	Increase	Small Change	Decrease
I	Decrease	Increase	Increase	Eliminate
D	Small Change	Decrease	Decrease	Small Change

Effects of increasing parameters

Ziegler-Nichols method Another tuning method is formally known as the *Ziegler-Nichols method*, introduced by John G. Ziegler and Nathaniel B. Nichols ². It starts in the same way as the method described before: first set the I and D gains to zero and then increase the P gain and expose the loop to external interference for example knocking the motor axis to cause it to move out of equilibrium in order to determine critical gain and period of oscillation until the output of the loop starts to oscillate. Write down the critical gain (K_c) and the oscillation period of the output (P_c). Then adjust the P, I and D controls as the table shows:

Control type	P	I	D
P	$.5K_c$		
PI	$.45K_c$	$P_c/1.2$	
PID	$.6K_c$	$P_c/2$	$P_c/8$

Final Steps After tuning the axis check the following error with Halscope to make sure it is within your machine requirements. More information on Halscope is in the HAL User manual.

²Introduced in the 1942 paper Optimum Settings for Automatic Controllers, DOI 10.1115/1.2899060 also available from [The Internet Archive](#).

Chapter 21

External Axis Offsets

External axis offsets are supported during teleop (world) jogs and coordinated (G-code) motion. External axis offsets are enabled on a per-axis basis by INI file settings and controlled dynamically by INI input pins. The INI interface is similar to that used for wheel jogging. This type of interface is typically implemented with a manual-pulse-generator (mpg) connected to an encoder INI component that counts pulses.

21.1 INI File Settings

For each axis letter (**L** in xyzabcuvw):

```
[AXIS_L]OFFSET_AV_RATIO = value (controls accel/vel for external offsets)
```

1. Allowed values: $0 \leq \text{value} \leq 0.9$
2. Disallowed values are replaced with 0.1 with message to stdout
3. Default value: 0 (disables external offset).
Consequence: omitted [**AXIS_L**]OFFSET_AV_RATIO disables external offset for the axis.
4. If nonzero, the OFFSET_AV_RATIO (**r**), adjusts the conventional (planning) max velocity and acceleration to preserve [**AXIS_L**] constraints:

```
planning max velocity      = (1-r) * MAX_VELOCITY
external offset velocity   = (  r) * MAX_VELOCITY

planning max acceleration  = (1-r) * MAX_ACCELERATION
external offset acceleration = (  r) * MAX_ACCELERATION
```

21.2 HAL Pins

21.2.1 Per-Axis Motion HAL Pins

For each axis letter (**L** in xyzabcuvw)

1. **axis.L.offset-enable** Input(bit): enable

2. **axis.L.eoffset-scale** Input(float): scale factor
3. **axis.L.eoffset-counts** Input(s32): input to counts register
4. **axis.L.eoffset-clear** Input(bit): clear requested offset
5. **axis.L.eoffset** Output(float): current external offset
6. **axis.L.eoffset-request** Output(float): requested external offset

21.2.2 Other Motion HAL Pins

1. **motion.eoffset-active** Output(bit): non-zero external offsets applied
2. **motion.eoffset-limited** Output(bit): motion inhibited due to soft limit

21.3 Usage

The axis input HAL pins (enable,scale,counts) are similar to the pins used for wheel jogging.

21.3.1 Offset Computation

At each servo period, the *axis.L.eoffset-counts* pin is compared to its value in the prior period. The increase or decrease (positive or negative delta) of the *axis.L.eoffset-counts* pin is multiplied by the current *axis.L.eoffset-scale* pin value. This product is accumulated in an internal register and exported to the *axis.L.eoffset-request* HAL pin. The accumulation register is reset to zero at each machine-on.

The requested offset value is used to plan the movement for the offset that is applied to the *L* coordinate and represented by the *axis.L.eoffset* HAL pin. The planned motion respects the allocated velocity and acceleration constraints and may be limited if the net motion (offset plus teleop jogging or coordinated motion) reaches a soft limit for the *L* coordinate.

For many applications, the *axis.L.eoffset-scale* pin is constant and the net *axis.L.eoffset-request* response to *axis.L.eoffset-counts* is equivalent to the product of the accumulated value of *axis.L.eoffset-counts* and the (constant) *axis.L.eoffset-scale* pin values.

21.3.2 Machine-off/Machine-on

When the machine is turned off, the **current position with external offsets is maintained** so that there is no unexpected motion at turn off or turn on.

At each startup (machine-on), the internal counts register for each HAL pin *axis.L.eoffset-counts* is zeroed and the corresponding HAL output pin *axis.L.eoffset* is reset to zero.

In other words, external offsets are **defined as ZERO at each startup** (machine-on) regardless of the value of the *axis.L.eoffset-counts* pins. To avoid confusion, it is recommended that all *axis.L.eoffset-counts* pins are set to zero when the machine is off.

21.3.3 Soft Limits

External axis offset movements are independently planned with velocity and acceleration settings specified by the `[AXIS_L]OFFSET_AV_RATIO`. The offsetting motion is not coordinated with teleop jogs nor with coordinated (G-code) motion. During teleop jogging and coordinated (G-code) motion, axis soft limits (`[AXIS_L]MIN_LIMIT,MAX_LIMIT`) restrict movement of the axis.

When external offsets are applied and motion reaches a soft limit (by external offset increases or teleop jogging or coordinated motion), the HAL pin `motion.eoffset-limited` is asserted and the axis value is held nominally to the soft limit. This HAL pin can be used by associated HAL logic to truncate additional eoffset counts or to stop the machine (connect to `halui.machine.off` for instance). If the axis is moved within the soft limit, the `motion.eoffset-limited` pin is reset.

When operating at a soft limit during coordinated motion that continues to change the planned axis value, the HAL output pin `axis.L.eoffset` will indicate the current offset — the distance needed to reach the limit instead of the computed offset request. This indicated value will change as the planned axis value changes.

The HAL pin `axis.L.eoffset-request` indicates the current requested offset that is the product of the internal counts register and the eoffset-scale. In general, the `axis.L.eoffset` pin value lags the `axis.L.eoffset-request` value since the external offset is subject to an acceleration limit. When operating at a soft limit, additional updates to the `axis.L.eoffset-counts` will continue to affect the requested external offset as reflected in the `axis.L.eoffset-request` HAL pin.

When teleop jogging with external offsets enabled **and** non-zero values applied, encountering a soft limit will stop motion in the offending axis **without a deceleration interval**. Similarly, during coordinated motion with external offsets enabled, reaching a soft limit will stop motion with no deceleration phase. For this case, it does not matter if the offsets are zero.

When motion is stopped with no deceleration phase, system **acceleration limits may be violated** and lead to: 1) a following error (and/or a thump) for a servo motor system, 2) a loss of steps for a stepper motor system. In general, it is recommended that external offsets are applied in a manner to avoid approaching soft limits.

21.3.4 Notes

External offsets apply to axis coordinate letters (xyzabcuvw). All joints must be homed before external axis offsets are honored.

If an `axis.L.eoffset-enable` HAL pin is reset when its offset is non-zero, the offset is maintained. The offset may be cleared by:

1. a *Machine-off/Machine on* toggle
2. reactivating the enable pin and incrementing/decrementing the `axis.L.eoffset-counts` HAL pin to return the offset to zero.
3. pulsing the `axis.L.eoffset-clear` HAL pin

External-offsets are intended for use with *small* offsets that are applied within the soft-limit bounds.

Soft limits are respected for both teleop jogging and coordinated motion when external offsets are applied. However, when a soft limit is reached during coordinated motion, reducing the offending external offset **may not move away** from the soft limit **if planned motion continues in the same direction**. This circumstance can occur since the rate of correcting offset removal (as set by `[AXIS_L]OFFSET_AV_RATIO`) may be less than the opposing planned rate of motion. In such cases, **pausing** (or stopping) the planned, coordinated motion will allow movement away from the soft limit when correcting changes are made in the offending external offset.

21.3.5 Warning

The use of external offsets can alter machine motion in a significant manner. The control of external offsets with HAL components and connections and any associated user interfaces should be carefully designed and tested before deployment.

21.4 Related HAL Components

21.4.1 eoffset_per_angle.comp

Component to compute an external offset from a function based on a measured angle (rotary coordinate or spindle). See the man page for details (**\$ man eoffset_per_angle**).

21.5 Testing

The external axis offset capability is enabled by adding an `[AXIS_L]` setting for each candidate axis. For example:

```
[AXIS_Z]
OFFSET_AV_RATIO = 0.2
```

For testing, it is convenient to simulate a jog wheel interface using the **sim_pin** GUI. For example, in a terminal:

```
$ sim_pin axis.z.eoffset-enable axis.z.eoffset-scale axis.z.eoffset-counts
```

The use of external offsets is aided by displaying information related to the current offsets: the current eoffset value and the requested eoffset value, the axis pos-cmd, and (for an identity kinematics machine) the corresponding joint motor pos-cmd and motor-offset. The provided sim configuration (see below) demonstrates an example PyVCP panel for the AXIS GUI.

In the absence of a custom display, **halshow** can be started as an auxiliary application with a custom watch list.

Example INI file settings to simulate the HAL pin eoffset connections and display eoffset information for the z axis (for identity kinematics with `z==joint2`):

```
[APPLICATIONS]
APP = sim_pin \
    axis.z.eoffset-enable \
    axis.z.eoffset-scale \
    axis.z.eoffset-counts \
    axis.z.eoffset-clear

APP = halshow --fformat "%0.5f" ./z.halshow
```

Where the file `z.halshow` (in the configuration directory) is:

```
pin+joint.2.motor-pos-cmd
pin+joint.2.motor-offset
pin+axis.z.pos-cmd
pin+axis.z.eoffset
pin+axis.z.eoffset-request
pin+motion.eoffset-limited
```

21.6 Examples

Provided simulation configurations demonstrate the use of external offsets in order to provide a starting point for user customization for real hardware.

The sim configurations utilize the INI setting `[HAL]HALFILE = LIB:basic_sim.tcl` to configure all routine HAL connections for the axes specified in the INI file `[TRAJ]COORDINATES=` setting. The HAL logic needed to demonstrate external offset functionality and the GUI HAL pin connections for a PyVCP panel are made in separate HAL files. A non-simulation configuration should replace the `LIB:basic_sim.tcl` item HALFILES appropriate to the machine. The provided PyVCP files (.hal and .xml) could be a starting point for application-specific GUI interfaces.

21.6.1 eoffsets.ini

The sim config `sim/configs/axis/external_offsets/eoffsets.ini` demonstrates a cartesian XYZ machine with controls to enable external offsets on any axis.

Displays are provided to show all important position and offset values.

A `sim_pin` GUI provides controls for the axis offset pins: `eoffset-scale` & `eoffset-counts` (via signal `e:<L>counts`), `eoffset-clear` (via signal `e:clearall`)

A script (`eoffsets_monitor.tcl`) is used to set `axis.L.counts` pins to zero at Machine-off.

21.6.2 jwp_z.ini

The sim config `sim/configs/axis/external_offsets/jwp_z.ini` demonstrates a jog-while-pause capability for a single (Z) coordinate:

Panel LEDs are provided to show important status items.

Controls are provided to set the `eoffset` scale factor and to increment/decrement/clear the `eoffset` counts.

21.6.3 dynamic_offsets.ini

This sim config `sim/configs/axis/external_offsets/dynamic_offsets.ini` demonstrates dynamically applied offsets by connecting a sinusoidal waveform to the z coordinate external offset inputs.

Panel LEDs are provided to show important status items.

Controls are provided to alter INI file settings for the Z axis max velocity and max acceleration.

Controls are provided to set the waveform generator parameters.

A `halscope` app is started to show the applied waveform, the offset response, and the motor cmd response.

Note

changes to the z coordinate max-acceleration and max-velocity are not acknowledged while a program is running.

21.6.4 opa.ini (eoffset_per_angle)

The opa.ini configuration uses the INI component eoffset_per_angle (**\$ man eoffset_per_angle**) to demonstrate an XZC machine with functional offsets computed from the C coordinate (angle) and applied to the transvers (X) coordinate. Offset computations are based on a specified reference radius typically set by a program (or MDI) M68 command to control a **motion.analog-out-NN** pin.

Panel LEDs are provided to show important status items.

Functions are provided for inside and outside polygons (nsides ≥ 3), sine waves and square waves. The functions can be multiplied in frequency using the fmul pin and modified in amplitude using the rfrac pin (fraction of reference radius).

Controls are provided to start/stop offset waveforms and to set the function type and its parameters.

Chapter 22

Tool Database Interface

Tool data is conventionally described by a tool table file specified by an inifile setting: [EMCIO]TOOL_TABLE. A tool table file consists of a text line for each available tool describing the tool's parameters, see [Tool Table Format](#).

The tool database interface provides an alternative method for obtaining tool data via a separate program that manages a database of tools.

22.1 Interface

22.1.1 INI file Settings

INI file settings enable the (optional) operation of a user-provided tool database program:

```
[EMCIO]
DB_PROGRAM = db_program
```

The optional **db_program** specifies the path to a user-provided executable program that provides tool data.

Note

INI file settings for [EMCIO]TOOL_TABLE are ignored when a **db_program** is specified.

Note

The **db_program** may be implemented in any language currently supported in LinuxCNC (e.g., bash scripts, python or tcl scripts, c/c++ programs) as long as it conforms to the interface messages received on stdin and replied on stdout. A **db_program** could manage data from a flat file, a relational database (SQLite for example), or other data sources.

22.1.2 db_program operation (v2.0)

When a **db_program** is specified, operation is as follows:

1. At startup, LinuxCNC starts the **db_program** and connects to its stdin and stdout.
-

2. The **db_program** must respond by writing a single line acknowledgement consisting of a version string (e.g., "v2.0"). No tools will be available if the version is not compatible with the LinuxCNC database interface version.
3. Upon a successful acknowledgement, LinuxCNC issues a *g* (**get**) command to request all tools. The **db_program** must respond with a sequence of replies to identify each available tool. The textual reply format is identical to the text line format used in conventional tool table files. A final response of "FINI" terminates the reply.
4. The **db_program** then enters an event wait loop to receive commands that indicate that tool data has been changed by LinuxCNC. Tool data changes include:
 - a) spindle loading(*Tn M6*)/unloading(*T0 M6*)
 - b) tool parameter changes (*G10L1Pn* for example)
 - c) tool substitutions (*M61Qn*).

When a tool data change occurs, LinuxCNC sends a command to the **db_program** consisting of an identifying command letter followed by a full or abbreviated tool data line. The **db_program** must respond with a reply to confirm receipt. If the reply includes the text "NAK", a message is printed to stdout but execution continues. The "NAK" message signifies a lack of synchronization between the **db_program** and LinuxCNC — accompanying text should give an indication for the cause of the fault.

The commands issued for tool data changes are:

- "*p*" put data changes caused by *G10L1*, *G10L10*, *G10L11* G-codes. The tool data line will include all elements of a tool table text line.
- "*l*" spindle_load (*TnM6*). The tool data line includes only the *T* and *P* items identifying the relevant tool number and pocket number.
- "*u*" spindle_unload (*T0M6*). The tool data line includes only the *T* and *P* items identifying the relevant tool number and pocket number

Note

When a NON_RANDOM tool changer is specified using [EMCIO]RANDOM_TOOL_CHANGER=0 (the default), the spindle_load command issued for *TnM6* (or *M61Qn*) is: *l Tn P0* (pocket 0 is the spindle). The spindle_unload command issued for *T0M6* is *u T0 P0*.

Note

When a RANDOM tool changer is specified using [EMCIO]RANDOM_TOOL_CHANGER=1, a pair of spindle_unload/spindle_load commands are issued at each tool exchange. The pair of commands issued for *TnM6* (or *M61Qn*) are *u Tu Pm* followed by *l Tn P0* where *u* is the current tool to be sent to pocket *m* and *n* is the new tool to load in the spindle (pocket 0). By convention, a tool number of 0 is used to specify an empty tool,

22.1.3 Usage

Using a **db_program** does not change the way LinuxCNC operates but provides support for new database functionality for tool management.

For example, a **db_program** database application can maintain the operating hours for all tools by tracking each load/unload of a tool. A machine could then have three 6 mm endmills in pockets 111, 112, and 113 with the database application programmed to assign tool number 110 to the 6 mm endmill with the fewest operating hours. Then, when a LinuxCNC program requests tool 110, the database would specify the appropriate pocket based on tool usage history.

Tool data changes made within LinuxCNC (*p,u,l* commands) are pushed immediately to the **db_program** which is expected to synchronize its source data. By default, LinuxCNC requests for tool data (*g* commands) are made at startup only. A database program may update tool usage data on a continuous basis so long-lived LinuxCNC applications may benefit by refreshing the tool data provided by the **db_program**. The G-code command **G10L0** can be used to request a tool data reload (*g* command) from within G-code programs or by MDI. A reload operation is also typically provided by a Graphical User Interface (GUI) so that on-demand reloads can be requested. For example, a python GUI application can use:

```
#!/usr/bin/env python3
from linuxcnc import command
command().load_tool_table()
```

Alternatively, a **db_program** may push its local data changes to synchronize its data with LinuxCNC by using the `load_tool_table()` interface command. Commands which push changes to LinuxCNC may be rejected if the interpreter is running. The interpreter state can be checked before issuing a `load_tool_table()` command. Example:

```
#!/usr/bin/env python3
import linuxcnc
s = linuxcnc.stat()
s.poll()
if s.interp_state == linuxcnc.INTERP_IDLE:
    linuxcnc.command().load_tool_table()
else: # defer until interp not idle
    ...
```

If the database application adds or removes tools after initialization, a call to `tooldb_tools()` must be issued with an updated `user_tools` list. The updated list of tools will be used on subsequent `get` commands or `load_tool_table()` requests.

Note

Removal of a tool number should only be done if the tool number is not currently loaded in spindle.

22.1.3.1 Debug Environmental Variables

Exporting the environmental variable `DB_SHOW` enables LinuxCNC prints (to stdout) that show tool data retrieved from the **db_program** at startup and at subsequent reloading of tool data.

Exporting the environmental variable `DB_DEBUG` enables LinuxCNC prints (to stdout) for additional debugging information about interface activity.

22.1.4 Example program

An example **db_program** (implemented as a python script) is provided with the simulation examples. The program demonstrates the required operations to:

1. acknowledge startup version
 2. receive tool data requests: *g* (**get** command)
 3. receive tool data updates: *p* (**put** command)
 4. receive tool load updates: *l* (**load_spindle** command)
 5. receive tool unload updates: *u* (**unload_spindle** command)
-

22.1.5 Python tooldb module

The example program uses a LinuxCNC provided python module (*tooldb*) that manages the low-level details for communication and version verification. This module uses callback functions specified by the **db_program** to respond to the *g* (get) command and the commands that indicate tool data changes (*p*, *l*, *u*).

The **db_program** uses the *tooldb* module by implementing the following python code:

```
user_tools = list(...)    # list of available tool numbers

def user_get_tool(toolno):
    # function to respond to 'g' (get) commands
    # called once for each toolno in user_tools
    ...
def user_put_tool(toolno,params):
    # function to respond to 'p' (put) commands
    ...
def user_load_spindle(toolno,params):
    # function to respond to 'l' (put) commands
    ...
def user_unload_spindle(toolno,params):
    # function to respond to 'u' (put) commands
    ...

#-----
# Begin:
from tooldb import tooldb_tools    # identify known tools
from tooldb import tooldb_callbacks # identify functions
from tooldb import tooldb_loop     # main loop

tooldb_tools(user_tools)
tooldb_callbacks(user_get_tool,
                 user_put_tool,
                 user_load_spindle,
                 user_unload_spindle,
                 )
tooldb_loop()
```

Note

Use of *tooldb* is not required — it is provided as a demonstration of the required interface and as a convenience for implementing python-based applications that interface with an external database.

22.2 Simulation configs

Simulation configs using the axis gui:

1. configs/sim/axis/db_demo/**db_ran**.ini (random_toolchanger)
2. configs/sim/axis/db_demo/**db_nonran**.ini (nonrandom_toolchanger)

Each sim config simulates a **db_program** implementing a database with 10 tools numbered 10—19.

The **db_program** is provided by a single script (db.py) and symbolic links to it for alternative uses: db_ran.py and db_nonran.py. (By default, the script implements random_toolchanger functionality. Nonrandom toolchanger functions are substituted if the link name includes the text "*nonran*").

The sim configs demonstrate the use of the python *tooldb* interface module and implement a basic flat-file database that tracks tool time usage for multiple tools having equal diameters. The database rules support selection of the tool having the lowest operating time.

The sim configs use a primary task to monitor and respond to tool updates initiated from within LinuxCNC. Separate, concurrent tasks show the code required when changes are initiated by the **db_program** and demonstrate methods for synchronizing LinuxCNC internal tooldata. Examples include:

1. updates of tool parameters
2. addition and removal of tool numbers

22.2.1 Notes

When a **db_program** is used in conjunction with a random tool changer ([EMCIO]RANDOM_TOOLCHANGE LinuxCNC maintains a file (*db_spindle.tbl* in the configuration directory) that consists of a single tool table line identifying the current tool in the spindle.

Chapter 23

Stand Alone Interpreter

The rs274 stand alone interpreter is available for use via the command line.

23.1 Usage

```
Usage: rs274 [-p interp.so] [-t tool.tbl] [-v var-file.var] [-n 0|1|2]
          [-b] [-s] [-g] [input file [output file]]

-p: Specify the pluggable interpreter to use
-t: Specify the .tbl (tool table) file to use
-v: Specify the .var (parameter) file to use
-n: Specify the continue mode:
    0: continue
    1: enter MDI mode
    2: stop (default)
-b: Toggle the 'block delete' flag (default: OFF)
-s: Toggle the 'print stack' flag (default: OFF)
-g: Toggle the 'go (batch mode)' flag (default: OFF)
-i: specify the .ini file (default: no ini file)
-T: call task_init()
-l: specify the log_level (default: -1)
```

23.2 Example

To see the output of a loop for example we can run rs274 on the following file and see that the loop never ends. To break out of the loop use Ctrl Z. The following two files are needed to run the example.

test.ngc

```
#<test> = 123.352

o101 while [[#<test> MOD 60 ] NE 0]
(debug,#<test>)
    #<test> = [#<test> + 1]
o101 endwhile

M2
```

test.tbl

```
T1 P1 Z0.511 D0.125 ;1/8 end mill  
T2 P2 Z0.1 D0.0625 ;1/16 end mill  
T3 P3 Z1.273 D0.201 ;#7 tap drill
```

command

```
rs274 -g test.ngc -t test.tbl
```

Part VI

Glossary

A listing of terms and what they mean. Some terms have a general meaning and several additional meanings for users, installers, and developers.

Acme Screw

A type of lead-screw that uses an Acme thread form. Acme threads have somewhat lower friction and wear than simple triangular threads, but ball-screws are lower yet. Most manual machine tools use acme lead-screws.

Axis

One of the computer controlled movable parts of the machine. For a typical vertical mill, the table is the X axis, the saddle is the Y axis, and the quill or knee is the Z axis. Angular axes like rotary tables are referred to as A, B, and C. Additional linear axes relative to the tool are called U, V, and W respectively.

Axis(GUI)

One of the Graphical User Interfaces available to users of LinuxCNC. It features the modern use of menus and mouse buttons while automating and hiding some of the more traditional LinuxCNC controls. It is the only open-source interface that displays the entire tool path as soon as a file is opened.

Gmoccapy (GUI)

A Graphical User Interfaces available to users of LinuxCNC. It features the use and feel of an industrial control and can be used with touch screen, mouse and keyboard. It support embedded tabs and hal driven user messages, it offers a lot of hal beens to be controlled with hardware. Gmoccapy is highly customizable.

Backlash

The amount of "play" or lost motion that occurs when direction is reversed in a lead screw. or other mechanical motion driving system. It can result from nuts that are loose on leadscrews, slippage in belts, cable slack, "wind-up" in rotary couplings, and other places where the mechanical system is not "tight". Backlash will result in inaccurate motion, or in the case of motion caused by external forces (think cutting tool pulling on the work piece) the result can be broken cutting tools. This can happen because of the sudden increase in chip load on the cutter as the work piece is pulled across the backlash distance by the cutting tool.

Backlash Compensation

Any technique that attempts to reduce the effect of backlash without actually removing it from the mechanical system. This is typically done in software in the controller. This can correct the final resting place of the part in motion but fails to solve problems related to direction changes while in motion (think circular interpolation) and motion that is caused when external forces (think cutting tool pulling on the work piece) are the source of the motion.

Ball Screw

A type of lead-screw that uses small hardened steel balls between the nut and screw to reduce friction. Ball-screws have very low friction and backlash, but are usually quite expensive.

Ball Nut

A special nut designed for use with a ball-screw. It contains an internal passage to re-circulate the balls from one end of the screw to the other.

CNC

Computer Numerical Control. The general term used to refer to computer control of machinery. Instead of a human operator turning cranks to move a cutting tool, CNC uses a computer and motors to move the tool, based on a part program.

Comp

A tool used to build, compile and install LinuxCNC HAL components.

Configuration(n)

A directory containing a set of configuration files. Custom configurations are normally saved in the users home/linuxcnc/configs directory. These files include LinuxCNC's traditional INI file and HAL files. A configuration may also contain several general files that describe tools, parameters, and NML connections.

Configuration(v)

The task of setting up LinuxCNC so that it matches the hardware on a machine tool.

Coordinate Measuring Machine

A Coordinate Measuring Machine is used to make many accurate measurements on parts. These machines can be used to create CAD data for parts where no drawings can be found, when a hand-made prototype needs to be digitized for moldmaking, or to check the accuracy of machined or molded parts.

Display units

The linear and angular units used for onscreen display.

DRO

A Digital Read Out is a system of position-measuring devices attached to the slides of a machine tool, which are connected to a numeric display showing the current location of the tool with respect to some reference position. DROs are very popular on hand-operated machine tools because they measure the true tool position without backlash, even if the machine has very loose Acme screws. Some DROs use linear quadrature encoders to pick up position information from the machine, and some use methods similar to a resolver which keeps rolling over.

EDM

EDM is a method of removing metal in hard or difficult to machine or tough metals, or where rotating tools would not be able to produce the desired shape in a cost-effective manner. An excellent example is rectangular punch dies, where sharp internal corners are desired. Milling operations can not give sharp internal corners with finite diameter tools. A *wire* EDM machine can make internal corners with a radius only slightly larger than the wire's radius. A *sinker* EDM can make internal corners with a radius only slightly larger than the radius on the corner of the sinking electrode.

EMC

The Enhanced Machine Controller. Initially a NIST project. Renamed to LinuxCNC in 2012.

EMCIO

The module within LinuxCNC that handles general purpose I/O, unrelated to the actual motion of the axes.

EMCMOT

The module within LinuxCNC that handles the actual motion of the cutting tool. It runs as a real-time program and directly controls the motors.

Encoder

A device to measure position. Usually a mechanical-optical device, which outputs a quadrature signal. The signal can be counted by special hardware, or directly by the parport with LinuxCNC.

Feed

Relatively slow, controlled motion of the tool used when making a cut.

Feed rate

The speed at which a cutting motion occurs. In auto or MDI mode, feed rate is commanded using an F word. F10 would mean ten machine units per minute.

Feedback

A method (e.g., quadrature encoder signals) by which LinuxCNC receives information about the position of motors

Feedrate Override

A manual, operator controlled change in the rate at which the tool moves while cutting. Often used to allow the operator to adjust for tools that are a little dull, or anything else that requires the feed rate to be "tweaked".

Floating Point Number

A number that has a decimal point. (12.300) In HAL it is known as float.

G-code

The generic term used to refer to the most common part programming language. There are several dialects of G-code, LinuxCNC uses RS274/NGC.

GUI

Graphical User Interface.

General

A type of interface that allows communications between a computer and a human (in most cases) via the manipulation of icons and other elements (widgets) on a computer screen.

LinuxCNC

An application that presents a graphical screen to the machine operator allowing manipulation of the machine and the corresponding controlling program.

HAL

Hardware Abstraction Layer. At the highest level, it is simply a way to allow a number of building blocks to be loaded and interconnected to assemble a complex system. Many of the building blocks are drivers for hardware devices. However, HAL can do more than just configure hardware drivers.

Home

A specific location in the machine's work envelope that is used to make sure the computer and the actual machine both agree on the tool position.

ini file

A text file that contains most of the information that configures LinuxCNC for a particular machine.

Instance

One can have an instance of a class or a particular object. The instance is the actual object created at runtime. In programmer jargon, the "Lassie" object is an instance of the "Dog" class.

Joint Coordinates

These specify the angles between the individual joints of the machine. See also Kinematics

Jog

Manually moving an axis of a machine. Jogging either moves the axis a fixed amount for each key-press, or moves the axis at a constant speed as long as you hold down the key. In manual mode, jog speed can be set from the graphical interface.

kernel-space

See real-time.

Kinematics

The position relationship between world coordinates and joint coordinates of a machine. There are two types of kinematics. Forward kinematics is used to calculate world coordinates from joint coordinates. Inverse kinematics is used for exactly the opposite purpose. Note that kinematics does not take into account, the forces, moments etc. on the machine. It is for positioning only.

Lead-screw

An screw that is rotated by a motor to move a table or other part of a machine. Lead-screws are usually either ball-screws or acme screws, although conventional triangular threaded screws may be used where accuracy and long life are not as important as low cost.

Machine units

The linear and angular units used for machine configuration. These units are specified and used in the ini file. HAL pins and parameters are also generally in machine units.

MDI

Manual Data Input. This is a mode of operation where the controller executes single lines of G-code as they are typed by the operator.

NIST

National Institute of Standards and Technology. An agency of the Department of Commerce in the United States.

NML

Neutral Message Language provides a mechanism for handling multiple types of messages in the same buffer as well as simplifying the interface for encoding and decoding buffers in neutral format and the configuration mechanism.

Offsets

An arbitrary amount, added to the value of something to make it equal to some desired value. For example, G-code programs are often written around some convenient point, such as X0, Y0. Fixture offsets can be used to shift the actual execution point of that G-code program to properly fit the true location of the vice and jaws. Tool offsets can be used to shift the "uncorrected" length of a tool to equal that tool's actual length.

Part Program

A description of a part, in a language that the controller can understand. For LinuxCNC, that language is RS-274/NGC, commonly known as G-code.

Program Units

The linear and angular units used in a part program. The linear program units do not have to be the same as the linear machine units. See G20 and G21 for more information. The angular program units are always measured in degrees.

Python

General-purpose, very high-level programming language. Used in LinuxCNC for the Axis GUI, the Stepconf configuration tool, and several G-code programming scripts.

Rapid

Fast, possibly less precise motion of the tool, commonly used to move between cuts. If the tool meets the workpiece or the fixturing during a rapid, it is probably a bad thing!

Rapid rate

The speed at which a rapid motion occurs. In auto or MDI mode, rapid rate is usually the maximum speed of the machine. It is often desirable to limit the rapid rate when testing a G-code program for the first time.

Real-time

Software that is intended to meet very strict timing deadlines. Under Linux, in order to meet these requirements it is necessary to install a realtime kernel such as RTAI and build the software to run in the special real-time environment. For this reason real-time software runs in kernel-space.

RTAI

Real Time Application Interface, see <https://www.rtai.org/>, the real-time extensions for Linux that LinuxCNC can use to achieve real-time performance.

RTLINUX

See <https://en.wikipedia.org/wiki/RTLinux>, an older real-time extension for Linux that LinuxCNC used to use to achieve real-time performance. Obsolete, replaced by RTAI.

RTAPI

A portable interface to real-time operating systems including RTAI and POSIX pthreads with realtime extensions.

RS-274/NGC

The formal name for the language used by LinuxCNC part programs.

Servo Motor

Generally, any motor that is used with error-sensing feedback to correct the position of an actuator. Also, a motor which is specially-designed to provide improved performance in such applications.

Servo Loop

A control loop used to control position or velocity of an motor equipped with a feedback device.

Signed Integer

A whole number that can have a positive or negative sign. In HAL it is known as s32. (A signed 32-bit integer has a usable range of -2,147,483,647 to +2,147,483,647.)

Spindle

The part of a machine tool that spins to do the cutting. On a mill or drill, the spindle holds the cutting tool. On a lathe, the spindle holds the workpiece.

Spindle Speed Override

A manual, operator controlled change in the rate at which the tool rotates while cutting. Often used to allow the operator to adjust for chatter caused by the cutter's teeth. Spindle Speed Override assumes that the LinuxCNC software has been configured to control spindle speed.

Stepconf

An LinuxCNC configuration wizard. It is able to handle many step-and-direction motion command based machines. It writes a full configuration after the user answers a few questions about the computer and machine that LinuxCNC is to run on.

Stepper Motor

A type of motor that turns in fixed steps. By counting steps, it is possible to determine how far the motor has turned. If the load exceeds the torque capability of the motor, it will skip one or more steps, causing position errors.

TASK

The module within LinuxCNC that coordinates the overall execution and interprets the part program.

Tcl/Tk

A scripting language and graphical widget toolkit with which several of LinuxCNCs GUIs and selection wizards were written.

Traverse Move

A move in a straight line from the start point to the end point.

Units

See "Machine Units", "Display Units", or "Program Units".

Unsigned Integer

A whole number that has no sign. In HAL it is known as u32. (An unsigned 32-bit integer has a usable range of zero to 4,294,967,296.)

World Coordinates

This is the absolute frame of reference. It gives coordinates in terms of a fixed reference frame that is attached to some point (generally the base) of the machine tool.

Part VII

Legal Section

Translations of this file provided in the source tree are not legally binding.

Chapter 24

Copyright Terms

Copyright (c) 2000-2022 LinuxCNC.org

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.1 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

Chapter 25

GNU Free Documentation License

GNU Free Documentation License Version 1.1, March 2000

Copyright © 2000 Free Software Foundation, Inc. 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA. Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other written document “free” in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondly, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of “copyleft”, which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. The “Document”, below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as “you”.

A “Modified Version” of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A “Secondary Section” is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document’s overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (For example, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The “Invariant Sections” are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License.

The “Cover Texts” are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, whose contents can be viewed and edited directly and straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup has been designed to thwart or discourage subsequent modification by readers is not Transparent. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML designed for human modification. Opaque formats include PostScript, PDF, proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies of the Document numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a publicly-accessible computer-network location containing a complete Transparent copy of the Document, free of added material, which the general network-using public has access to download anonymously at no charge using public-standard network protocols. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

- A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
- B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has less than five).
- C. State on the Title page the name of the publisher of the Modified Version, as the publisher.
- D. Preserve all the copyright notices of the Document.
- E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
- F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
- G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
- H. Include an unaltered copy of this License.
- I. Preserve the section entitled "History", and its title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
- J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
- K. In any section entitled "Acknowledgements" or "Dedications", preserve the section's title, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
- L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
- M. Delete any section entitled "Endorsements". Such a section may not be included in the Modified Version.
- N. Do not retitle any existing section as "Endorsements" or to conflict in title with any Invariant Section.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the

Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections entitled "History" in the various original documents, forming one section entitled "History"; likewise combine any sections entitled "Acknowledgements", and any sections entitled "Dedications". You must delete all sections entitled "Endorsements."

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, does not as a whole count as a Modified Version of the Document, provided no compilation copyright is claimed for the compilation. Such a compilation is called an "aggregate", and this License does not apply to the other self-contained works thus compiled with the Document, on account of their being thus compiled, if they are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one quarter of the entire aggregate, the Document's Cover Texts may be placed on covers that surround only the Document within the aggregate. Otherwise they must appear on covers around the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License provided that you also include the original English version of this License. In case of a disagreement between the translation and the original English version of this License, the original English version will prevail.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See <http://www.gnu.org/copyleft/>.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option

of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

Copyright (c) YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.1 or any later version published by the Free Software Foundation; with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST. A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have no Invariant Sections, write "with no Invariant Sections" instead of saying which ones are invariant. If you have no Front-Cover Texts, write "no Front-Cover Texts" instead of "Front-Cover Texts being LIST"; likewise for Back-Cover Texts.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

Chapter 26

Index

/ Block Delete, [351](#)
 [APPLICATIONS] Section, [492](#)
 [AXIS_<letter>] Sections, [495](#)
 [DISPLAY] Section, [483](#)
 [EMCIO] Section, [503](#)
 [EMCMOT] Section, [490](#)
 [EMC] Section, [482](#)
 [FILTER] Section, [487](#)
 [HALUI] Section, [492](#)
 [HAL] Section, [490](#)
 [JOINT_<num>] Sections, [496](#)
 [RS274NGC] Section, [488](#)
 [SPINDLE_<num>] Section(s), [502](#)
 [TASK] Section, [490](#)
 [TRAJ] Section, [492](#)
 0-10 Volt Spindle Speed Example, [927](#)
 5-Axis Kinematics, [1141](#)
 7i65, [1056](#)

A

abs, [1054](#)
 acme screw, [1184](#)
 addf, [986](#)
 and2, [992](#), [1053](#)
 ANGULAR UNITS, [494](#)
 Arc Distance Mode, [415](#)
 at_pid, [1057](#)
 Automatic Login, [82](#)
 axes, [28](#)
 AXIS
 Keyboard Shortcuts, [127](#)
 Tool Touch Off, [118](#)
 axis, [1053](#), [1184](#)
 Axis Configuration, [39](#), [44](#)
 AXIS GUI, [114](#)
 axis-interface, [904](#)
 AXIS: axisui pins, [138](#)
 AXIS: Preview Control, [137](#)
 AXIS: Virtual Control Panel, [137](#)
 AXIS:.axisrc, [136](#)
 AXIS:Jogwheel, [136](#)
 AXIS:Lathe Mode, [131](#)

AXIS:Manual Tool Change, [130](#)
 AXIS:Manual Toolchange Window, [130](#)
 AXIS:Python Modules, [130](#)

B

Backlash, [497](#)
 backlash, [1184](#)
 backlash compensation, [1184](#)
 ball nut, [1184](#)
 ball screw, [1184](#)
 Base Period Maximum Jitter, [40](#)
 Binary Operators, [361](#)
 biquad, [1055](#)
 bit, [991](#)
 bldc_hall3, [1057](#)
 blend, [1054](#)

C

Calling Files, [435](#)
 Canned Cycle Errors, [404](#)
 Cartesian machines, [1113](#)
 cd, [84](#)
 Changing Directories, [84](#)
 charge_pump, [1057](#)
 CL Programming, [937](#)
 clarke2, [1057](#)
 clarke3, [1057](#)
 clarkeinv, [1057](#)
 ClassicLadder, [1053](#)
 ClassicLadder Programming
 CL Programming, [937](#)
 CNC, [979](#), [1184](#)
 Comment Parameters, [368](#)
 Comments, [366](#), [430](#), [480](#)
 comp, [1054](#), [1184](#)
 Compensation, [497](#)
 Compensation End Point, [454](#)
 Complete Machine Configuration, [51](#)
 Components, [479](#)
 Comments, [480](#)
 Custom sections and variables, [481](#)
 Include, [482](#)
 Sections, [480](#)

- Variables, [481](#)
- Conditional Loops, [434](#)
- Configuration Launcher, [33](#)
- Configuration Selection, [27](#)
- connecting-rs485, [916](#)
- constant, [1054](#)
- Controlled Point, [29](#)
- conv_bit_s32, [1055](#)
- conv_bit_u32, [1055](#)
- conv_float_s32, [1055](#)
- conv_float_u32, [1055](#)
- conv_s32_bit, [1055](#)
- conv_s32_float, [1055](#)
- conv_s32_u32, [1055](#)
- conv_u32_bit, [1055](#)
- conv_u32_float, [1055](#)
- conv_u32_s32, [1055](#)
- coolant, [28](#)
- Cooling, [30](#)
- coordinate measuring machine, [1185](#)
- Coordinate Systems, [343](#)
- Core components, [1025](#)
- counter, [1054](#)
- Creating Userspace Python Components, [1103](#)
- Custom sections and variables, [481](#)
- Cutter Radius Compensation, [453](#)

D

- dbounce, [1053](#)
- ddt, [1054](#)
- deadzone, [1054](#)
- debounce, [1053](#), [1078](#)
- Debug Messages, [368](#)
- Determining Spindle Calibration, [48](#)
- DH parameters Examples, [1119](#)
- display units, [1185](#)
- Distance to accelerate to max speed, [45](#)
- Driver Microstepping, [44](#)
- Driver Type, [40](#)
- DRO, [1185](#)
- Dwell
 - Feed Out, [414](#)
- dwll, [30](#)

E

- edge, [1053](#)
- EDM, [1185](#)
- EMC, [1185](#)
- EMCIO, [1185](#)
- EMCMOT, [1185](#)
- encoder, [1057](#), [1071](#), [1185](#)
- Encoder Block Diagram, [1072](#)
- encoder_ratio, [1057](#)
- estop_latch, [1057](#)
- Expressions, [361](#)
- externaloffsets, [1170](#)

F

- F: Set Feed Rate, [436](#)
- feed, [1185](#)
- Feed and Speed Interaction, [32](#)
- Feed Out, [413](#), [414](#)
- feed override, [29](#)
- Feed Rate, [30](#)
- feed rate, [1185](#)
- feedback, [1185](#)
- feedcomp, [1058](#)
- feedrate override, [1186](#)
- FERROR, [498](#)
- File Requirements, [368](#)
- File Size, [369](#)
- Finding Maximum Acceleration, [47](#)
- Finding Maximum Velocity, [46](#)
- Five Phase, [1068](#)
- flipflop, [1053](#)
- float, [991](#)
- Four Phase, [1067](#)
- Functions, [361](#)

G

- G-code, [22](#), [1186](#)
- G-code Best Practices, [370](#)
- G-code Order of Execution, [369](#)
- G-Code Table, [372](#)
- G0 Rapid Move, [373](#)
- G1 Linear Move, [373](#)
- G10 L0 Reload Tool Table Data, [383](#)
- G10 L1 Tool Table, [383](#)
- G10 L10 Set Tool Table, [385](#)
- G10 L11 Set Tool Table, [386](#)
- G10 L2 Coordinate System, [384](#)
- G10 L20 Set Coordinate System, [386](#)
- G17 - G19.1 Plane Select, [387](#)
- G2
 - G3 Arc Move, [374](#)
- G20 Units, [387](#)
- G28 Go/Set Predefined Position, [387](#)
- G3 Arc Move, [374](#)
- G30 Go/Set Predefined Position, [388](#)
- G33 Spindle Synchronized Motion, [388](#)
- G33.1 Rigid Tapping, [389](#)
- G38.n Probe, [390](#)
- G4 Dwell, [379](#)
- G40 Cutter Compensation Off, [392](#)
- G41 G42 Cutter Compensation, [392](#)
- G41.1 G42.1 Dynamic Compensation, [393](#)
- G43 Tool Length Offset, [393](#)
- G43.1 Dynamic Tool Length Offset, [394](#)
- G43.2 Apply additional Tool Length Offset, [394](#)
- G49 Cancel Tool Length Offset, [395](#)
- G5 Cubic spline, [379](#)
- G5.1 Quadratic spline, [380](#)
- G5.2 G5.3 NURBS Block, [381](#)
- G53 Machine Coordinates, [395](#)
- G54-G59.3 Select Coordinate System, [395](#)

G61 Exact Path Mode, [396](#)
G61.1 Exact Stop Mode, [396](#)
G64 Path Blending, [396](#)
G7 Lathe Diameter Mode, [382](#)
G70 Lathe finishing cycle, [397](#)
G71 G72 Lathe roughing cycles, [398](#)
G73 Drilling Cycle with Chip Break, [399](#)
G74 Left-hand Tapping Cycle with Dwell, [399](#)
G76 Threading Cycle, [400](#)
G8 Lathe Radius Mode, [382](#)
G80 Cancel Modal Motion, [406](#)
G80-G89 Canned Cycles, [403](#)
G81 Drilling Cycle, [407](#)
G82 Drilling Cycle Dwell, [412](#)
G83 Peck Drilling, [412](#)
G84 Right-hand Tapping Cycle Dwell, [413](#)
G85 Boring
 Feed Out, [413](#)
G86 Boring
 Spindle Stop
 Rapid Move Out, [414](#)
G87 Back Boring Cycle, [414](#)
G88 Boring Cycle
 Spindle Stop
 Manual Out, [414](#)
G89 Boring
 Dwell
 Feed Out, [414](#)
G90
 G91 Distance Mode, [414](#)
G91 Distance Mode, [414](#)
G92 Coordinate System Offset, [415](#)
G92.1
 G92.2 Reset G92 Offsets, [416](#)
G92.2 Reset G92 Offsets, [416](#)
G92.3 Restore G92 Offsets, [416](#)
G93
 G94
 G95 Feed Rate Mode, [416](#)
G94
 G95 Feed Rate Mode, [416](#)
G95 Feed Rate Mode, [416](#)
G96
 G97 Spindle Control Mode, [417](#)
G97 Spindle Control Mode, [417](#)
G98
 G99 Canned Cycle Return, [417](#)
G99 Canned Cycle Return, [417](#)
gantrykins, [1056](#)
gearchange, [1058](#)
genhexkins, [1056](#)
genserkins, [1056](#)
gladevc, [1053](#)
GladeVCP: Glade Virtual Control Panel, [604](#)
Graphical User Interfaces, [11](#)
GS2 VFD Driver, [858](#)
GUI, [1184](#), [1186](#)

H

HAL, [979](#), [1186](#)
HAL addf
 addf, [986](#)
HAL Analog Input, [1033](#)
HAL Analog Input Functions, [1034](#)
HAL Analog Input Parameters, [1034](#)
HAL Analog Input Pins, [1034](#)
HAL Analog Output, [1034](#)
HAL Analog Output Functions, [1035](#)
HAL Analog Output Parameters, [1035](#)
HAL Analog Output Pins, [1034](#)
HAL and2
 and2, [992](#)
HAL Basics, [985](#)
HAL Basics Summary, [981](#)
HAL Bit
 bit, [991](#)
HAL Canonical Device Interfaces, [1032](#)
HAL Commands, [985](#)
HAL Component, [982](#)
HAL Component Generator, [1087](#)
HAL Component List, [1051](#)
HAL Components, [984](#)
HAL Concepts, [982](#)
HAL Conversion Components, [994](#)
HAL Data, [991](#)
HAL Digital Input, [1032](#)
HAL Digital Input Functions, [1033](#)
HAL Digital Input Parameters, [1033](#)
HAL Digital Input Pins, [1033](#)
HAL Digital Output, [1033](#)
HAL Digital Output Functions, [1033](#)
HAL Digital Output Parameters, [1033](#)
HAL Digital Output Pins, [1033](#)
HAL Entity Names, [1023](#)
HAL Examples, [1081](#)
HAL Files, [991](#)
HAL Float
 float, [991](#)
HAL General Naming Conventions, [1023](#)
HAL General Reference, [1023](#)
HAL Hardware Driver Function Names, [1025](#)
HAL Hardware Driver Naming Conventions, [1023](#)
HAL Hardware Driver Pins/Parameters Names,
 [1024](#)
HAL Introduction, [979](#)
HAL loadrt
 loadrt, [986](#)
HAL loadusr
 loadusr, [987](#)
HAL Logic Components, [992](#)
HAL Logic Examples, [994](#)
HAL net
 net, [988](#)
HAL not
 not, [993](#)

- HAL or2
 - or2, [993](#)
- HAL Parameter, [983](#)
- HAL Parameters, [992](#)
- HAL Part selection, [980](#)
- HAL Physical Pin, [983](#)
- HAL Pin, [983](#)
- HAL pins and INI values, [360](#)
- HAL s32
 - s32, [991](#)
- HAL setp
 - setp, [989](#)
- HAL sets
 - sets, [990](#)
- HAL Signal, [983](#)
- Hal stepgen Functions, [1069](#)
- HAL stepgen parameters, [1064](#)
- HAL stepgen pins, [1063](#)
- HAL stepgen Step Types, [1064](#)
- HAL System Design, [979](#)
- HAL time, [992](#)
- HAL Timing Issues, [984](#)
- HAL tmax, [992](#)
- HAL Tools, [1035](#)
- HAL Tutorial, [998](#)
- HAL Type, [983](#)
- HAL u32
 - u32, [991](#)
- HAL unlinkp
 - unlinkp, [990](#)
- HAL weighted_sum
 - weighted_sum, [994](#)
- HAL xor2
 - xor2, [994](#)
- HAL: Implementation, [981](#)
- HAL: Interconnections Design, [981](#)
- HAL: Testing, [981](#)
- HAL:Function, [984](#)
- HAL:Thread, [984](#)
- HAL:Velocity example, [1084](#)
- hal_ppmc, [1056](#)
- Halcmd Tutorial, [998](#)
- Halmeter, [1036](#)
 - Tutorial Halmeter, [1004](#)
- Halshow, [1044](#)
- Halshow: Watch Tab, [1050](#)
- Halui Examples, [816](#)
- History, [5](#)
- hm2_7i43, [1056](#)
- hm2_pci, [1056](#)
- HOME, [509](#)
- home, [1186](#)
- HOME ABSOLUTE ENCODER, [510](#)
- HOME IGNORE LIMITS, [509](#)
- HOME INDEX NO ENCODER RESET, [509](#)
- HOME IS SHARED, [510](#)
- Home Latch Direction, [45](#)

- HOME LATCH VEL, [508](#)
- Home Location, [45](#)
- HOME OFFSET, [509](#)
- HOME SEARCH VEL, [498](#), [508](#)
- Home Search Velocity, [45](#)
- HOME SEQUENCE, [510](#)
- Home Switch Location, [45](#)
- HOME USE INDEX, [509](#)
- hostmot2, [1056](#)
- hypot, [1054](#)

I

- ilowpass, [1058](#)
- Immediate Homing, [511](#)
- Include, [482](#)
- Indirection, [435](#)
- Inhibiting Homing, [512](#)
- INI, [1186](#)
- INI Configuration, [479](#)
- INI File
 - Components, [479](#)
 - Comments, [480](#)
 - Custom sections and variables, [481](#)
 - Include, [482](#)
 - Sections, [480](#)
 - Variables, [481](#)
- Sections, [482](#)
 - [APPLICATIONS] Section, [492](#)
 - [AXIS_<letter>] Sections, [495](#)
 - [DISPLAY] Section, [483](#)
 - [EMCIO] Section, [503](#)
 - [EMCMOT] Section, [490](#)
 - [EMC] Section, [482](#)
 - [FILTER] Section, [487](#)
 - [HALUI] Section, [492](#)
 - [HAL] Section, [490](#)
 - [JOINT_<num>] Sections, [496](#)
 - [RS274NGC] Section, [488](#)
 - [SPINDLE_<num>] Section(s), [502](#)
 - [TASK] Section, [490](#)
 - [TRAJ] Section, [492](#)
 - KINS Section, [495](#)
- INI settings (HAL pins), [1031](#)
- Instance, [1186](#)
- integ, [1054](#)
- invert, [1054](#)
- iocontrol (HAL pins), [1031](#)

J

- jog, [1186](#)
- joint coordinates, [1186](#)
- joyhandle, [1058](#)

K

- Keyboard Shortcuts, [127](#)
- kinematics, [1113](#), [1186](#)
- kins, [1056](#)

KINS Section, [495](#)
knob2float, [1058](#)

L

Latency Test, [40](#), [467](#)
Latency Testing, [467](#)
Lathe Tool Orientation, [87](#)
Lathe Tool Table, [87](#)
Lathe User Information, [86](#)
lead screw, [1186](#)
Leadscrew Pitch, [44](#)
limit1, [1055](#)
limit2, [1055](#)
limit3, [1055](#)
Line Number, [351](#)
LINEAR UNITS, [494](#)
Linux FAQ, [82](#)
LinuxCNC User Introduction
 User Introduction, [9](#)
loadrt, [986](#)
loadusr, [987](#)
Local Offsets, [395](#)
LOCKING INDEXER, [511](#)
Logging, [367](#)
logic, [1053](#)
loop, [1188](#)
Looping, [433](#)
lowpass, [1055](#)
lut5, [1053](#), [1080](#)

M

M0

M1 Program Pause, [418](#)

M0 Mandatory Program Pause, [418](#)

M1 Optional Program Pause, [418](#)

M1 Program Pause, [418](#)

M100-M199 User Defined Commands, [428](#)

M19 Orient Spindle, [421](#)

M2 Program End, [418](#)

M3

M4

M5 Spindle Control, [419](#)

M30 Pallet Exchange and Program End, [418](#)

M4

M5 Spindle Control, [419](#)

M48

M49 Speed and Feed Override Control, [422](#)

M49 Speed and Feed Override Control, [422](#)

M5 Spindle Control, [419](#)

M50 Feed Override Control, [422](#)

M51 Spindle Speed Override, [422](#)

M52 Adaptive Feed Control, [422](#)

M53 Feed Stop Control, [422](#)

M6-Tool-Change, [420](#)

M60 Pallet Change Pause, [419](#)

M61 Set Current Tool, [423](#)

M62 - M65 Digital Output Control, [423](#)

M66 Wait on Input, [423](#)

M67 Analog Output

 Synchronized, [424](#)

M68 Analog Output, [425](#)

M7

M8

M9 Coolant Control, [421](#)

M70 Save Modal State, [425](#)

M71 Invalidate Stored Modal State, [426](#)

M72 Restore Modal State, [426](#)

M73 Save and Autorestore Modal State, [427](#)

M8

M9 Coolant Control, [421](#)

M9 Coolant Control, [421](#)

M98

M99, [432](#)

M99, [432](#)

Machine Name, [39](#)

Machine Overview, [28](#)

Machine Units, [39](#)

machine units, [1187](#)

maj3, [1055](#)

Man Pages, [83](#)

Manual Out, [414](#)

match8, [1053](#)

MAX ACCELERATION, [494](#)

MAX LIMIT, [495](#), [497](#)

Max Step Rate, [40](#)

MAX VELOCITY, [494](#)

Maximum Acceleration, [45](#)

Maximum Velocity, [44](#)

maxkins, [1056](#)

MDI, [1187](#)

mdro GUI, [339](#)

Messages, [367](#)

Min Base Period, [40](#)

MIN ERROR, [498](#)

MIN LIMIT, [495](#), [497](#)

minmax, [1058](#)

Modal Groups, [365](#)

Modal Groups: G-codes, [365](#)

Modal Groups: M-codes, [366](#)

Motion, [1026](#)

motion, [1053](#)

motion (HAL pins), [1027](#)

Motor Steps Per Revolution, [44](#)

Moveoff, [559](#)

mult2, [1054](#)

mux16, [1054](#)

mux2, [1054](#)

mux4, [1054](#)

mux8, [1054](#)

N

Named Parameters, [356](#)

near, [1054](#)

net, [988](#)

NGCGUI, [192](#)

NIST, [1187](#)

NML, [1187](#)

not, [993](#), [1053](#)

Numbered Parameters, [354](#)

Numbers, [352](#)

O

O Codes, [430](#)

O-Code Errors, [436](#)

offset, [1054](#)

offsets, [1187](#)

oneshot, [1053](#)

Operating without Home Switches, [52](#)

Operators Precedence, [361](#)

or2, [993](#), [1053](#)

ORIENT OFFSET, [488](#)

Other Codes, [436](#)

P

Panelui, [801](#)

PARAMETER FILE, [488](#)

Parameters, [32](#), [353](#)

Parport block diag, [853](#)

part Program, [1187](#)

Path Control Mode, [31](#)

pci-card connectors, [901](#)

PID, [1074](#)

pid, [1057](#)

PID Block Diagram, [1075](#)

PID Tuning, [1167](#)

pin-numbering-axis, [903](#)

pin-numbering-endsw, [913](#)

pin-numbering-gpio, [901](#)

Plasma Cutting Primer, [95](#)

pluto-servo pinout, [892](#)

pluto-step, [893](#)

pluto-step pinout, [894](#)

pluto-step timings, [895](#)

pluto_servo, [1056](#)

pluto_step, [1056](#)

Polar Coordinates, [363](#)

Predefined Named Parameters, [357](#)

Print Messages, [368](#)

Probe Logging, [367](#)

program units, [1187](#)

Pulley Ratio, [44](#)

Pulse rate at max speed, [45](#)

pumakins, [1057](#)

PWM Spindle Speed Example, [927](#)

PWMgen, [1069](#)

pwmgen, [1057](#)

PyVCP Widgets Reference, [572](#)

PyVCP with AXIS, [570](#)

Q

QtVcp Overview, [672](#)

R

rapid, [1187](#)

Rapid Move Out, [414](#)

rapid rate, [1187](#)

real-time, [1187](#)

refsig-timing-diagram, [909](#)

Repeat Loop, [435](#)

Return Values, [436](#)

rotatekins, [1057](#)

RS274NGC, [1188](#)

RS274NGC STARTUP CODE, [488](#)

RTAI, [1187](#)

RTAPI, [1188](#)

RTLINUX, [1187](#)

Running LinuxCNC, [33](#)

S

s32, [991](#)

S: Set Spindle Speed, [436](#)

sample_hold, [1058](#)

sampler, [1058](#)

scale, [1055](#)

scarakins, [1057](#)

Sections, [480](#), [482](#)

[APPLICATIONS] Section, [492](#)

[AXIS_<letter>] Sections, [495](#)

[DISPLAY] Section, [483](#)

[EMCIO] Section, [503](#)

[EMCMOT] Section, [490](#)

[EMC] Section, [482](#)

[FILTER] Section, [487](#)

[HALUI] Section, [492](#)

[HAL] Section, [490](#)

[JOINT_<num>] Sections, [496](#)

[RS274NGC] Section, [488](#)

[SPINDLE_<num>] Section(s), [502](#)

[TASK] Section, [490](#)

[TRAJ] Section, [492](#)

KINS Section, [495](#)

select8, [1053](#)

serport, [1056](#)

servo motor, [1188](#)

setp, [989](#)

sets, [990](#)

Sherline, [477](#)

Siggen, [1079](#)

siggen, [1058](#)

Signed Integer, [1188](#)

sim_encoder, [1058](#)

Simulated Encoder, [1077](#)

Soft Limits, [122](#)

sphereprobe, [1058](#)

spindle, [28](#), [1188](#)

spindle (HAL pins), [1029](#)

Spindle At Speed Example, [930](#)

Spindle Direction Example, [928](#)

Spindle Enable Example, [927](#)

spindle override, [29](#)
Spindle Soft Start Example, [928](#)
Spindle Speed Control, [48](#)
Spindle Stop
 Manual Out, [414](#)
 Rapid Move Out, [414](#)
Spindle Synchronized Motion Example, [929](#)
Spindle-synchronized motion, [48](#)
Starting LinuxCNC, [26](#)
stepgen, [1057](#), [1061](#)
Stepgen Block Diagram, [1062](#)
stepgen Example, [1007](#)
Stepper Configuration, [564](#)
Stepper Configuration Wizard, [37](#)
Stepper Diagnostics, [474](#)
stepper motor, [1188](#)
steptest, [1058](#)
streamer, [1058](#)
Subroutine Parameters, [356](#)
SUBROUTINE PATH, [488](#)
Subroutines, [431](#)
 Conditional Loops, [434](#)
 Looping, [433](#)
 M98
 M99, [432](#)
 Repeat Loop, [435](#)
sum2, [1054](#)
supply, [1058](#)
Synchronized, [424](#)
System Parameters, [358](#)

T
T: Select Tool, [437](#)
Table Travel, [45](#)
TASK, [1188](#)
Test this axis, [45](#)
threads, [1053](#)
threadtest, [1058](#)
time, [1058](#)
Time to accelerate to max speed, [45](#)
timedelay, [1058](#)
timedelta, [1058](#)
Tk, [1188](#)
TkLinuxCNC Common Keyboard Shortcuts, [232](#)
TkLinuxCNC Interpreter, [230](#)
toggle, [1059](#)
toggle2nist, [1058](#)
Tool Changers, [452](#)
Tool Compensation, [448](#)
Tool IO, [451](#)
Tool Length Compensation, [453](#)
Tool Table, [32](#), [449](#)
Tool Table Format, [449](#)
Tool Touch Off, [118](#)
torch height control, [1056](#)
Touch Off, [448](#)
touchygui, [210](#)

TP, [20](#)
Trajectory Control, [20](#), [396](#)
Trajectory Planning
 TP, [20](#)
Trajectory Planning:Path Following, [20](#)
Trajectory Planning:Planning Moves, [22](#)
Trajectory Planning:Programming the Planner,
 [21](#)
Traverse Move, [1188](#)
tripodkins, [1056](#)
tristate_bit, [1059](#)
tristate_float, [1059](#)
Trivial Kinematics, [1113](#)
trivkins, [1056](#)
Tutorial Halmeter, [1004](#)
Tutorial Halscope, [1011](#)
Two and Three Phase, [1066](#)

U
u32, [991](#)
Unary operations, [361](#)
UNITS, [497](#)
units, [30](#), [1188](#)
unlinkp, [990](#)
Unsigned Integer, [1188](#)
updown, [1054](#)
User Concepts, [20](#)
User Foreword, [8](#)
User Introduction, [9](#)
USER M PATH, [488](#)

V
Variables, [481](#)
VOLATILE HOME, [511](#)

W
watchdog, [1059](#)
wcomp, [1054](#)
weighted_sum, [994](#), [1055](#)
Words, [352](#)
world coordinates, [1188](#)

X
xor2, [994](#), [1053](#)
Xylotex, [477](#)