Integrator Manual V2.7.0-pre4, 2015-02-21

Contents

I LinuxCNC Introduction 1

1 Integrator Concepts 3

1.1 Stepper Systems . . . 3
1.1.1 Base Period . . . 3
1.1.2 Step Timing . . . 3
1.2 Servo Systems . . . 4
1.2.1 Basic Operation . . . 4
1.2.2 Proportional term . . . 5
1.2.3 Integral term . . . 5
1.2.4 Derivative term . . . 5
1.2.5 Loop tuning . . . 6
1.2.6 Manual tuning . . . 6
1.3 RTAI . . . 6
1.3.1 ACPI . . . 6

II Configuration 7

2 Latency Test 8

3 Starting LinuxCNC 12

3.1 Running LinuxCNC . . . 12
3.2 Files Used for Configuration . . . 13
3.3 TWOPASS . . . 14

4 INI Configuration 16

4.1 The INI File Components . . . 16
4.1.1 Comments . . . 16
4.1.2 Sections . . . 17
4.1.3 Variables . . . 17

4.1.4 Custom Sections and Variables . . . 18
4.1.5 Include Files . . . 18
4.2 INI File Sections . . . 19
4.2.1 [EMC] Section . . . 19
4.2.2 [DISPLAY] Section . . . 19
4.2.3 [FILTER] Section . . . 21
4.2.4 [RS274NGC] Section . . . 22
4.2.5 [EMCMOT] Section . . . 23
4.2.6 [TASK] Section . . . 23
4.2.7 [HAL] section . . . 23
4.2.8 [HALUI] section . . . 24
4.2.9 [APPLICATIONS] Section . . . 24
4.2.10 [TRAJ] Section . . . 25
4.2.11 [AXIS_<num>] Section . . . 27
4.2.11.1 Homing . . . 29
4.2.11.2 Servo . . . 29
4.2.11.3 Stepper . . . 32
4.2.12 [EMCIO] Section . . . 32

5 Homing Configuration **34**
5.1 Overview . . . 34
5.2 Homing Sequence . . . 34
5.3 Configuration . . . 36
5.3.1 HOME_SEARCH_VEL . . . 36
5.3.2 HOME_LATCH_VEL . . . 36
5.3.3 HOME_FINAL_VEL . . . 36
5.3.4 HOME_IGNORE_LIMITS . . . 36
5.3.5 HOME_USE_INDEX . . . 37
5.3.6 HOME_OFFSET . . . 37
5.3.7 HOME . . . 37
5.3.8 HOME_IS_SHARED . . . 37
5.3.9 HOME_SEQUENCE . . . 37
5.3.10 VOLATILE_HOME . . . 37
5.3.11 LOCKING_INDEXER . . . 37
5.3.12 Immediate Homing . . . 38

6 Lathe Configuration **39**
6.1 Default Plane . . . 39
6.2 INI Settings . . . 39

7 HALTCL Files **40**
7.1 Compatibility . . . 40
7.2 Haltcl Commands . . . 40
7.3 Haltcl Inifile variables . . . 41
7.4 Converting .hal files to .tcl files . . . 41
7.5 Haltcl Notes . . . 41
7.6 Haltcl Examples . . . 42
7.7 Haltcl Interactive . . . 42
7.8 Haltcl Distribution Examples (sim) . . . 42

8 Core Components **43**
8.1 Motion . . . 43
8.1.1 Options . . . 44
8.1.2 Pins . . . 44
8.1.2.1 HAL pin usage for M19 orient spindle . . . 45
8.1.3 Parameters . . . 46
8.1.4 Functions . . . 47
8.2 Axis (Joints) . . . 47
8.2.1 Pins . . . 47
8.2.2 Parameters . . . 48
8.3 iocontrol . . . 48
8.3.1 Pins . . . 48
8.4 ini settings . . . 49
8.4.1 Pins . . . 49

9 Stepper Configuration **50**
9.1 Introduction . . . 50
9.2 Maximum step rate . . . 50
9.3 Pinout . . . 50
9.3.1 standard_pinout.hal . . . 51
9.3.2 Overview . . . 52
9.3.3 Changing the standard_pinout.hal . . . 52
9.3.4 Changing polarity of a signal . . . 53
9.3.5 Adding PWM Spindle Speed Control . . . 53
9.3.6 Adding an enable signal . . . 53
9.3.7 External ESTOP button . . . 53

10 Basic HAL Reference **55**
10.1 HAL Commands . . . 55
10.1.1 loadrt . . . 56
10.1.2 addf . . . 56
10.1.3 loadusr . . . 57
10.1.4 net . . . 57
10.1.5 setp . . . 58
10.1.6 sets . . . 59
10.1.7 unlinkp . . . 59
10.1.8 Obsolete Commands . . . 59
10.1.8.1 linksp . . . 59
10.1.8.2 linkps . . . 60
10.1.8.3 newsig . . . 60
10.2 HAL Data . . . 60
10.2.1 Bit . . . 60
10.2.2 Float . . . 60
10.2.3 s32 . . . 60
10.2.4 u32 . . . 60
10.3 HAL Files . . . 61
10.4 HAL Components . . . 61
10.5 Logic Components . . . 61
10.5.1 and2 . . . 61
10.5.2 not . . . 62
10.5.3 or2 . . . 62
10.5.4 xor2 . . . 62
10.5.5 Logic Examples . . . 63
10.6 Conversion Components . . . 63
10.6.1 weighted_sum . . . 63

11 Extending LinuxCNC **65**
11.1 Introduction: Extending the RS274NGC Interpreter by Remapping Codes . . . 65
11.1.1 A Definition: Remapping Codes . . . 65
11.1.2 Why would you want to extend the RS274NGC Interpreter? . . . 65
11.1.2.1 How to glue things together . . . 66
11.1.2.2 How Embedded Python fits in . . . 66
11.1.2.3 A Word on Embedded Python . . . 66
11.2 Getting started . . . 66
11.2.1 Picking a code . . . 67
11.2.2 Parameter handling . . . 67

11.2.3 Handling results . . . 67
11.2.4 Execution sequencing . . . 67
11.2.5 An minimal example remapped code . . . 68
11.3 Configuring Remapping . . . 68
11.3.1 The REMAP statement . . . 68
11.3.2 Useful REMAP option combinations . . . 69
11.3.3 The argspec parameter . . . 69
11.3.3.1 Example for named parameter passing to NGC procedures . . . 70
11.3.3.2 Example for positional parameter passing to NGC procedures . . . 71
11.3.3.3 Simple example for named parameter passing to a Python function . . . 71
11.3.3.4 Advanced example: Remapped codes in pure Python . . . 71
11.4 Upgrading an existing configuration for remapping . . . 73
11.5 Remapping tool change-related codes: T, M6, M61 . . . 73
11.5.1 Overview . . . 73
11.5.2 Understanding the role of iocontrol with remapped tool change codes . . . 74
11.5.3 Specifying the M6 replacement . . . 75
11.5.4 Configuring iocontrol with a remapped M6 . . . 76
11.5.5 Writing the change and prepare O-word procedures . . . 76
11.5.6 Making minimal changes to the built in codes, including `M6` . . . 77
11.5.7 Specifying the T (prepare) replacement . . . 77
11.5.8 Error handling: dealing with abort . . . 78
11.5.9 Error handling: failing a remapped code NGC procedure . . . 79
11.6 Remapping other existing codes: S, M0, M1, M60 . . . 80
11.6.1 Automatic gear selection be remapping S (set spindle speed) . . . 80
11.6.2 Adjusting the behavior of M0, M1, M60 . . . 80
11.7 Creating new G-code cycles . . . 80
11.8 Configuring Embedded Python . . . 81
11.8.1 Python plugin : ini file configuration . . . 81
11.8.2 Executing Python statements from the interpreter . . . 81
11.9 Programming Embedded Python in the RS274NGC Interpreter . . . 82
11.9.1 The Python plugin namespace . . . 82
11.9.2 The Interpreter as seen from Python . . . 82
11.9.3 The Interpreter `__init__` and `__delete__` functions . . . 82
11.9.4 Calling conventions: NGC to Python . . . 83
11.9.4.1 Calling O-word Python subroutines . . . 83
11.9.4.2 Return values of O-word Python subroutines . . . 83
11.9.4.3 Calling conventions for *prolog=* and *epilog=* subroutines . . . 84
11.9.4.4 Calling conventions for *python=* subroutines . . . 84
11.9.4.5 Dealing with queue-buster: Probe, Tool change and waiting for a HAL pin . . . 85

11.9.5 Calling conventions: Python to NGC . . . 85
11.9.5.1 Inserting parameters in a prolog, and retrieving them in an epilog . . . 85
11.9.5.2 Calling the interpreter from Python . . . 86
11.9.5.3 Interpreter Exception during execute() . . . 86
11.9.5.4 Canon . . . 87
11.9.6 Built in modules . . . 87
11.10 Adding Predefined Named Parameters . . . 87
11.11 Standard Glue routines . . . 88
11.11.1 T: prepare_prolog and prepare_epilog . . . 88
11.11.1.1 Actions of prepare_prolog . . . 88
11.11.1.2 Actions of prepare_epilog . . . 89
11.11.2 M6: change_prolog and change_epilog . . . 89
11.11.2.1 Actions of change_prolog . . . 89
11.11.2.2 Actions of change_epilog . . . 89
11.11.3 G code Cycles: cycle_prolog and cycle_epilog . . . 90
11.11.3.1 Actions of cycle_prolog . . . 90
11.11.3.2 Actions of cycle_epilog . . . 90
11.11.4 S (Set Speed) : setspeed_prolog and setspeed_epilog . . . 90
11.11.5 F (Set Feed) : setfeed_prolog and setfeed_epilog . . . 90
11.11.6 M61 Set tool number : settool_prolog and settool_epilog . . . 90
11.12 Remapped code execution . . . 91
11.12.1 NGC procedure call environment during remaps . . . 91
11.12.2 Nested remapped codes . . . 91
11.12.3 Sequence number during remaps . . . 91
11.12.4 Debugging flags . . . 91
11.12.5 Debugging Embedded Python code . . . 91
11.13 Axis Preview and Remapped code execution . . . 92
11.14 Remappable Codes . . . 93
11.14.1 Existing codes which can be remapped . . . 93
11.14.2 Currently unallocated G-codes: . . . 93
11.14.3 Currently unallocated M-codes: . . . 94
11.14.4 readahead time and execution time . . . 94
11.14.5 plugin/pickle hack . . . 95
11.14.6 Module, methods, classes, etc reference . . . 95
11.15 Introduction: Extending Task Execution . . . 95
11.15.1 Why would you want to change Task Execution? . . . 95
11.15.2 A diagram: task, interp, iocontrol, UI (??) . . . 95
11.16 Models of Task execution . . . 95
11.16.1 Traditional iocontrol/iocontrolv2 execution . . . 95

11.16.2 Redefining IO procedures . . . 95
11.16.3 Execution-time Python procedures . . . 95
11.17 A short survey of LinuxCNC program execution . . . 95
11.17.1 Interpreter state . . . 96
11.17.2 Task and Interpreter interaction, Queuing and Read-Ahead . . . 96
11.17.3 Predicting the machine position . . . 96
11.17.4 Queue-busters break position prediction . . . 96
11.17.5 How queue-busters are dealt with . . . 97
11.17.6 Word order and execution order . . . 97
11.17.7 Parsing . . . 97
11.17.8 Execution . . . 98
11.17.9 Procedure execution . . . 98
11.17.10 How tool change currently works . . . 98
11.17.10.1 How tool information is communicated . . . 98
11.17.11 How Tx (Prepare Tool) works . . . 99
11.17.11.1 Interpreter action on a Tx command . . . 99
11.17.11.2 Task action on SELECT_POCKET . . . 99
11.17.11.3 Iocontrol action on EMC_TOOL_PREPARE . . . 99
11.17.11.4 Building the prolog and epilog for Tx . . . 99
11.17.12 How M6 (Change tool) works . . . 99
11.17.12.1 Interpreter action on a M6 command . . . 99
11.17.12.2 What task does when it sees a CHANGE_TOOL command . . . 100
11.17.12.3 Iocontrol action on EMC_TOOL_LOAD . . . 100
11.17.12.4 Building the prolog and epilog for M6 . . . 100
11.17.13 How M61 (Change tool number) works . . . 100
11.17.13.1 Building the replacement for M61 . . . 100
11.18 Optional Interpreter features: ini file configuration . . . 101
11.19 Named parameters and inifile variables . . . 101
11.20 Named parameters and HAL items . . . 102
11.21 Status . . . 102
11.22 Build notes - Lucid (10.04) . . . 103
11.23 Build notes - Hardy (8.04) . . . 103
11.24 Workarounds . . . 104
11.25 Changes . . . 104

12 Moveoff Component **105**

12.1 Modifying an existing configuration . . . 105

III GUI 107

13 Python Virtual Control Panel 108

13.1 Introduction . . . 108
13.2 Panel Construction . . . 109
13.3 Security . . . 110
13.4 AXIS . . . 110
13.5 Stand Alone . . . 111
13.6 Widgets . . . 112
13.6.1 Syntax . . . 112
13.6.2 General Notes . . . 112
13.6.2.1 Comments . . . 113
13.6.2.2 Editing the XML file . . . 113
13.6.2.3 Colors . . . 113
13.6.2.4 HAL Pins . . . 113
13.6.3 Label . . . 114
13.6.4 Multi_Label . . . 114
13.6.5 LEDs . . . 114
13.6.5.1 Round LED . . . 115
13.6.5.2 Rectangle LED . . . 115
13.6.6 Buttons . . . 115
13.6.6.1 Text Button . . . 116
13.6.6.2 Checkbutton . . . 116
13.6.6.3 Radiobutton . . . 116
13.6.7 Number Displays . . . 117
13.6.7.1 Number . . . 117
13.6.7.2 s32 Number . . . 118
13.6.7.3 u32 Number . . . 118
13.6.7.4 Bar . . . 118
13.6.7.5 Meter . . . 118
13.6.8 Number Inputs . . . 119
13.6.8.1 Spinbox . . . 119
13.6.8.2 Scale . . . 120
13.6.8.3 Dial . . . 121
13.6.8.4 Jogwheel . . . 121
13.6.9 Images . . . 122
13.6.9.1 Image Bit . . . 122
13.6.9.2 Image u32 . . . 122
13.6.10 Containers . . . 123

13.6.10.1 Borders . . . 123
13.6.10.2 Hbox . . . 124
13.6.10.3 Vbox . . . 124
13.6.10.4 Labelframe . . . 125
13.6.10.5 Table . . . 125
13.6.10.6 Tabs . . . 126

14 PyVCP Examples **128**
14.1 AXIS . . . 128
14.2 Floating . . . 128
14.3 Jog Buttons . . . 129
14.3.1 Create the Widgets . . . 130
14.3.2 Make Connections . . . 132
14.4 Port Tester . . . 132
14.5 GS2 RPM Meter . . . 135
14.5.1 The Panel . . . 135
14.5.2 The Connections . . . 137

15 Glade Virtual Control Panel **138**
15.1 What is GladeVCP? . . . 138
15.1.1 PyVCP versus GladeVCP at a glance . . . 138
15.2 A Quick Tour with the Example Panel . . . 139
15.2.1 Exploring the example panel . . . 142
15.2.2 Exploring the User Interface description . . . 142
15.2.3 Exploring the Python callback . . . 143
15.3 Creating and Integrating a Glade user interface . . . 143
15.3.1 Prerequisite: Glade installation . . . 143
15.3.2 Running Glade to create a new user interface . . . 143
15.3.3 Testing a panel . . . 144
15.3.4 Preparing the HAL command file . . . 144
15.3.5 Integrating into Axis like PyVCP . . . 145
15.3.6 Integrating into Axis as a tab next to DRO and Preview . . . 145
15.3.7 Integrating into Touchy . . . 146
15.4 GladeVCP command line options . . . 146
15.5 Understanding the gladeVCP startup process . . . 147
15.6 HAL Widget reference . . . 148
15.6.1 Widget and HAL pin naming . . . 148
15.6.2 Python attributes and methods of HAL Widgets . . . 149
15.6.3 Setting pin and widget values . . . 149

15.6.4 The hal-pin-changed signal . . . 149
15.6.5 Buttons . . . 150
15.6.6 Scales . . . 151
15.6.7 SpinButton . . . 151
15.6.8 Hal_Dial . . . 151
15.6.9 Jog Wheel . . . 153
15.6.10 Label . . . 155
15.6.11 Containers: HAL_HideTable HAL_Table State_Sensitive_Table and HAL_HBox . . . 155
15.6.12 LED . . . 156
15.6.13 ProgressBar . . . 156
15.6.14 ComboBox . . . 157
15.6.15 Bars . . . 158
15.6.16 Meter . . . 159
15.6.17 Gremlin tool path preview for .ngc files . . . 159
15.6.18 HAL_Offset . . . 162
15.6.19 DRO widget . . . 162
15.6.20 Combi_DRO widget . . . 163
15.6.21 IconView (File selection) widget . . . 166
15.6.22 Calculator widget . . . 169
15.6.23 Tooleditor widget . . . 170
15.6.24 Offsetpage . . . 170
15.6.25 HAL_sourceview widget . . . 172
15.6.26 MDI history . . . 173
15.6.27 Animated function diagrams: HAL widgets in a bitmap . . . 173
15.7 Action Widgets reference . . . 174
15.7.1 EMC Action widgets . . . 175
15.7.2 EMC ToggleAction widgets . . . 175
15.7.3 The Action_MDI Toggle and Action_MDI widgets . . . 175
15.7.4 A simple example: Execute MDI command on button press . . . 175
15.7.5 Parameter passing with Action_MDI and ToggleAction_MDI widgets . . . 176
15.7.6 An advanced example: Feeding parameters to an O-word subroutine . . . 176
15.7.7 Preparing for an MDI Action, and cleaning up afterwards . . . 177
15.7.8 Using the LinuxCNC Stat object to deal with status changes . . . 177
15.8 GladeVCP Programming . . . 178
15.8.1 User Defined Actions . . . 178
15.8.2 An example: adding custom user callbacks in Python . . . 179
15.8.3 HAL value change events . . . 179
15.8.4 Programming model . . . 179
15.8.4.1 The simple handler model . . . 180

15.8.4.2 The class-based handler model . . . 180
15.8.4.3 The get_handlers protocol . . . 180
15.8.5 Initialization sequence . . . 181
15.8.6 Multiple callbacks with the same name . . . 181
15.8.7 The GladeVCP -U <useropts> flag . . . 181
15.8.8 Persistent variables in GladeVCP . . . 182
15.8.8.1 Persistence, program versions and the signature check . . . 182
15.8.9 Using persistent variables . . . 182
15.8.10 Saving the state on Gladvcp shutdown . . . 183
15.8.11 Saving state when Ctrl-C is pressed . . . 183
15.8.12 Hand-editing .ini files . . . 184
15.8.13 Adding HAL pins . . . 184
15.8.14 Adding timers . . . 184
15.8.15 Setting HAL widget properties programmatically . . . 184
15.8.16 Examples, and rolling your own GladeVCP application . . . 185
15.9 FAQ . . . 185
15.10 Troubleshooting . . . 186
15.11 Implementation note: Key handling in Axis . . . 186
15.12 Adding Custom Widgets . . . 186

16 HAL User Interface **187**
16.1 Introduction . . . 187
16.2 Halui pin reference . . . 187

17 Halui Examples **193**
17.1 Remote Start . . . 193
17.2 Pause & Resume . . . 194

IV Hardware Drivers **195**

18 Parallel Port Driver **196**
18.1 Parport . . . 196
18.1.1 Installing . . . 196
18.1.2 Pins . . . 198
18.1.3 Parameters . . . 199
18.1.4 Functions . . . 199
18.1.5 Common problems . . . 199
18.1.6 Using DoubleStep . . . 199

19 AX5214H Driver **201**
19.1 Installing . . . 201
19.2 Pins . . . 201
19.3 Parameters . . . 201
19.4 Functions . . . 202

20 GS2 VFD Driver **203**
20.1 Command Line Options . . . 203
20.2 Pins . . . 203
20.3 Parameters . . . 204

21 Mesa HostMot2 Driver **205**
21.1 Introduction . . . 205
21.2 Firmware Binaries . . . 205
21.3 Installing Firmware . . . 206
21.4 Loading HostMot2 . . . 206
21.5 Watchdog . . . 206
21.5.1 Pins: . . . 206
21.5.2 Parameters: . . . 206
21.6 HostMot2 Functions . . . 207
21.7 Pinouts . . . 207
21.8 PIN Files . . . 208
21.9 Firmware . . . 208
21.10 HAL Pins . . . 208
21.11 Configurations . . . 209
21.12 GPIO . . . 211
21.12.1 Pins . . . 211
21.12.2 Parameters . . . 211
21.13 StepGen . . . 212
21.13.1 Pins . . . 212
21.13.2 Parameters . . . 212
21.13.3 Output Parameters . . . 213
21.14 PWMGen . . . 213
21.14.1 Pins . . . 213
21.14.2 Parameters . . . 213
21.14.3 Output Parameters . . . 214
21.15 Encoder . . . 214
21.15.1 Pins . . . 214
21.15.2 Parameters . . . 215

21.16 5i25 Configuration . . . 215
21.16.1 Firmware . . . 215
21.16.2 Configuration . . . 215
21.16.3 SSERIAL Configuration . . . 216
21.16.4 7i77 Limits . . . 216
21.17 Example Configurations . . . 216

22 Motenc Driver **217**
22.1 Pins . . . 217
22.2 Parameters . . . 218
22.3 Functions . . . 218

23 Opto22 Driver **219**
23.1 The Adapter Card . . . 219
23.2 The Driver . . . 219
23.3 Pins . . . 219
23.4 Parameters . . . 220
23.5 FUNCTIONS . . . 220
23.6 Configuring I/O Ports . . . 220
23.7 Pin Numbering . . . 221

24 Pico Drivers **222**
24.1 Command Line Options . . . 222
24.2 Pins . . . 223
24.3 Parameters . . . 224
24.4 Functions . . . 225

25 Pluto P Driver **226**
25.1 General Info . . . 226
25.1.1 Requirements . . . 226
25.1.2 Connectors . . . 226
25.1.3 Physical Pins . . . 226
25.1.4 LED . . . 227
25.1.5 Power . . . 227
25.1.6 PC interface . . . 227
25.1.7 Rebuilding the FPGA firmware . . . 227
25.1.8 For more information . . . 227
25.2 Pluto Servo . . . 227
25.2.1 Pinout . . . 228
25.2.2 Input latching and output updating . . . 229

25.2.3 HAL Functions, Pins and Parameters . . . 229
25.2.4 Compatible driver hardware . . . 230
25.3 Pluto Step . . . 230
25.3.1 Pinout . . . 230
25.3.2 Input latching and output updating . . . 231
25.3.3 Step Waveform Timings . . . 231
25.3.4 HAL Functions, Pins and Parameters . . . 232

26 Servo To Go Driver **233**
26.1 Installing . . . 233
26.2 Pins . . . 234
26.3 Parameters . . . 234
26.4 Functions . . . 234

27 ShuttleXpress **235**
27.1 Description . . . 235
27.2 Setup . . . 235
27.3 Pins . . . 235

28 General Mechatronics Driver **237**
28.1 I/O connectors . . . 238
28.1.1 Pins . . . 239
28.1.2 Parameters . . . 239
28.2 Axis connectors . . . 240
28.2.1 Axis interface modules . . . 240
28.2.2 Encoder . . . 241
28.2.2.1 Pins . . . 242
28.2.2.2 Parameters . . . 242
28.2.2.3 HAL example . . . 243
28.2.3 Stepgen module . . . 243
28.2.3.1 Pins . . . 245
28.2.3.2 Parameters . . . 245
28.2.3.3 HAL example . . . 246
28.2.4 Enable and Fault signals . . . 247
28.2.4.1 Pins . . . 247
28.2.5 Axis DAC . . . 247
28.2.5.1 Pins . . . 248
28.2.5.2 Parameters . . . 248
28.3 CAN-bus servo amplifiers . . . 248
28.3.1 Pins . . . 250

28.3.2 Parameters . . . 250
28.4 Watchdog timer . . . 250
28.4.1 Pins . . . 250
28.4.2 Parameters . . . 250
28.5 End-, homing- and E-stop switches . . . 251
28.5.1 Pins . . . 252
28.5.2 Parameters . . . 252
28.6 Status LEDs . . . 252
28.6.1 CAN . . . 252
28.6.2 RS485 . . . 252
28.6.3 EMC . . . 253
28.6.4 Boot . . . 253
28.6.5 Error . . . 253
28.7 RS485 I/O expander modules . . . 253
28.7.1 Relay output module . . . 254
28.7.1.1 Pins . . . 254
28.7.1.2 Parameters . . . 254
28.7.1.3 HAL example . . . 255
28.7.2 Digital input module . . . 255
28.7.2.1 Pins . . . 255
28.7.2.2 HAL example . . . 255
28.7.3 DAC & ADC module . . . 255
28.7.3.1 Pins . . . 256
28.7.3.2 Parameters . . . 256
28.7.3.3 HAL example . . . 256
28.7.4 Teach Pendant module . . . 257
28.7.4.1 Pins . . . 257
28.7.4.2 Parameters . . . 257
28.7.4.3 HAL example . . . 257
28.8 Errata . . . 258
28.8.1 GM6-PCI card Errata . . . 258
28.8.1.1 Rev. 1.2 . . . 258

V Advanced Topics 259

29 Python Interface 260

29.1 The linuxcnc Python module . . . 260
29.2 Usage Patterns for the LinuxCNC NML interface . . . 260
29.3 Reading LinuxCNC status . . . 261

29.3.1 linuxcnc.stat attributes . . . 261
29.3.2 The axis dictionary . . . 265
29.4 Preparing to send commands . . . 266
29.5 Sending commands through `linuxcnc.command` . . . 267
29.5.1 `linuxcnc.command` attributes . . . 268
29.5.2 `linuxcnc.command` methods: . . . 268
29.6 Reading the error channel . . . 270
29.7 Reading ini file values . . . 270
29.8 The `linuxcnc.positionlogger` type . . . 271
29.8.1 members . . . 271
29.8.2 methods . . . 271

30 Kinematics **272**
30.1 Introduction . . . 272
30.1.1 Joints vs. Axes . . . 272
30.2 Trivial Kinematics . . . 272
30.3 Non-trivial kinematics . . . 273
30.3.1 Forward transformation . . . 274
30.3.2 Inverse transformation . . . 274
30.4 Implementation details . . . 275

31 Stepper Tuning **276**
31.1 Getting the most out of Software Stepping . . . 276
31.1.1 Run a Latency Test . . . 276
31.1.2 Figure out what your drives expect . . . 277
31.1.3 Choose your BASE_PERIOD . . . 277
31.1.4 Use steplen, stepspace, dirsetup, and/or dirhold . . . 278
31.1.5 No Guessing! . . . 278

32 PID Tuning **279**
32.1 PID Controller . . . 279
32.1.1 Control loop basics . . . 279
32.1.2 Theory . . . 280
32.1.2.1 Proportional . . . 280
32.1.2.2 Integral . . . 280
32.1.2.3 Derivative . . . 280
32.1.3 Loop Tuning . . . 280
32.1.3.1 Simple method . . . 281
32.1.3.2 Ziegler-Nichols method . . . 281
32.1.3.3 Final Steps . . . 281

VI Ladder Logic **282**

33 Classicladder Introduction **283**

33.1 History . . . 283
33.2 Introduction . . . 283
33.3 Example . . . 284
33.4 Basic Latching On-Off Circuit . . . 284

34 Classicladder Programming **286**

34.1 Ladder Concepts . . . 286
34.2 Languages . . . 286
34.3 Components . . . 286
34.3.1 Files . . . 287
34.3.2 Realtime Module . . . 287
34.3.3 Variables . . . 287
34.4 Loading the Classic Ladder user module . . . 288
34.5 Classic Ladder GUI . . . 288
34.5.1 Sections Manager . . . 289
34.5.2 Section Display . . . 289
34.5.3 The Variable Windows . . . 290
34.5.4 Symbol Window . . . 293
34.5.5 The Editor window . . . 294
34.5.6 Config Window . . . 295
34.6 Ladder objects . . . 297
34.6.1 CONTACTS . . . 297
34.6.2 IEC TIMERS . . . 297
34.6.3 TIMERS . . . 298
34.6.4 MONOSTABLES . . . 298
34.6.5 COUNTERS . . . 298
34.6.6 COMPARE . . . 299
34.6.7 VARIABLE ASSIGNMENT . . . 300
34.6.8 COILS . . . 301
34.6.8.1 JUMP COIL . . . 302
34.6.8.2 CALL COIL . . . 302
34.7 Classic Ladder Variables . . . 302
34.8 GRAFCET Programming . . . 303
34.9 Modbus . . . 304
34.9.1 MODBUS Settings . . . 307
34.9.2 MODBUS Info . . . 308

34.9.3 Communication Errors . . . 308
34.9.4 MODBUS Bugs . . . 308
34.10 Setting up Classic Ladder . . . 309
34.10.1 Add the Modules . . . 309
34.10.2 Adding Ladder Logic . . . 309

35 Classicladder Examples **316**
35.1 Wrapping Counter . . . 316
35.2 Reject Extra Pulses . . . 317
35.3 External E-Stop . . . 318
35.4 Timer/Operate Example . . . 321

VII Hardware Examples **323**

36 PCI Parallel Port **324**

37 Spindle Control **325**
37.1 0-10v Spindle Speed . . . 325
37.2 PWM Spindle Speed . . . 325
37.3 Spindle Enable . . . 326
37.4 Spindle Direction . . . 326
37.5 Spindle Soft Start . . . 326
37.6 Spindle Feedback . . . 327
37.6.1 Spindle Synchronized Motion . . . 327
37.6.2 Spindle At Speed . . . 328

38 MPG Pendant **329**

39 GS2 Spindle **332**

VIII Diagnostics **333**

40 Stepper Diagnostics **334**
40.1 Common Problems . . . 334
40.1.1 Stepper Moves One Step . . . 334
40.1.2 No Steppers Move . . . 334
40.1.3 Distance Not Correct . . . 334
40.2 Error Messages . . . 334
40.2.1 Following Error . . . 334
40.2.2 RTAPI Error . . . 335
40.3 Testing . . . 335
40.3.1 Step Timing . . . 335

41 Glossary **337**

42 Legal Section **342**

42.1 Copyright Terms 342

42.2 GNU Free Documentation License 342

43 Index **346**

The LinuxCNC Team

Part I

LinuxCNC 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.

Chapter 1

Integrator Concepts

1.1 Stepper Systems

1.1.1 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.

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 us direction signal hold time requirement, and your latency test said you have a maximum latency of 11 us , then if you set the BASE_PERIOD to 20+11 = 31 us 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 us hold time requirement. That plus the 11 us latency is what forces us to use a slow 31 us 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 us hold time requirement, then the next longest time is the 4.5 us high time. Add the 11 us latency to the 4.5 us high time, and you get a minimum period of 15.5 us . When you try 15.5 us , you find that the computer is sluggish, so you settle on 16 us . If we leave *dirhold* at 1 (the default), then the minimum time between step and direction is the 16 us period minus the 11 us latency = 5 us , which is not enough. We need another 15 us . Since the period is 16 us , 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 us , 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 of the HAL manual.

1.1.2 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.

[1] This 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.

[2] steplen 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.

[3] dirhold refers to a parameter that adjusts the length of the direction hold time.

1.2 Servo Systems

1.2.1 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.

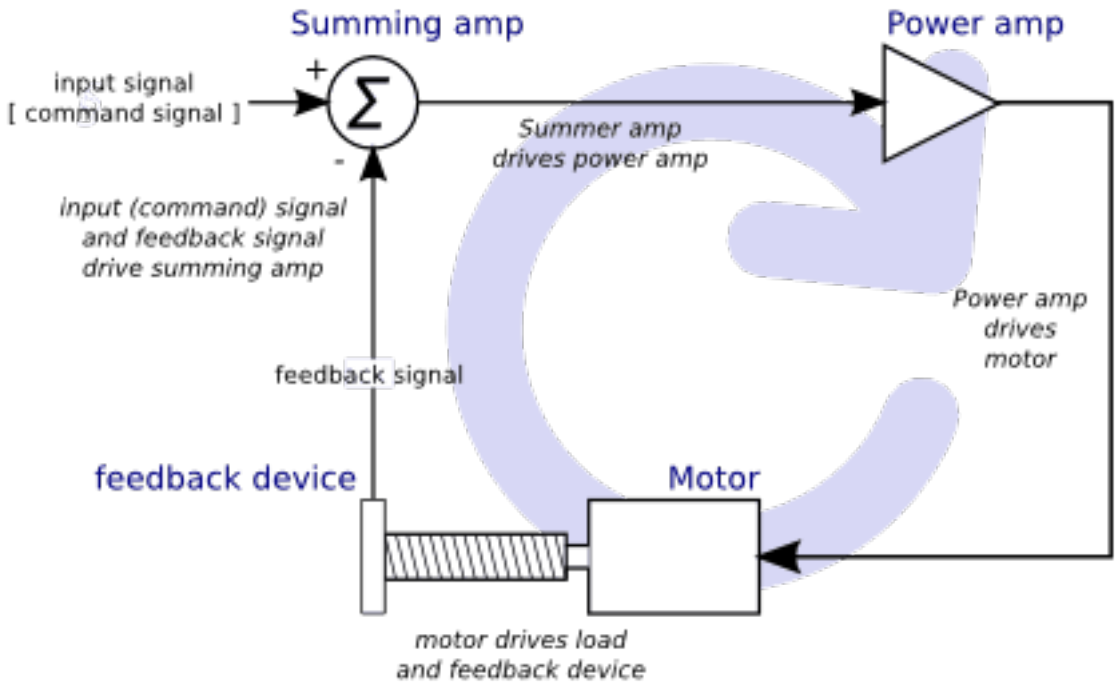


Figure 1.1: Servo Loop

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.

[4] If 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.

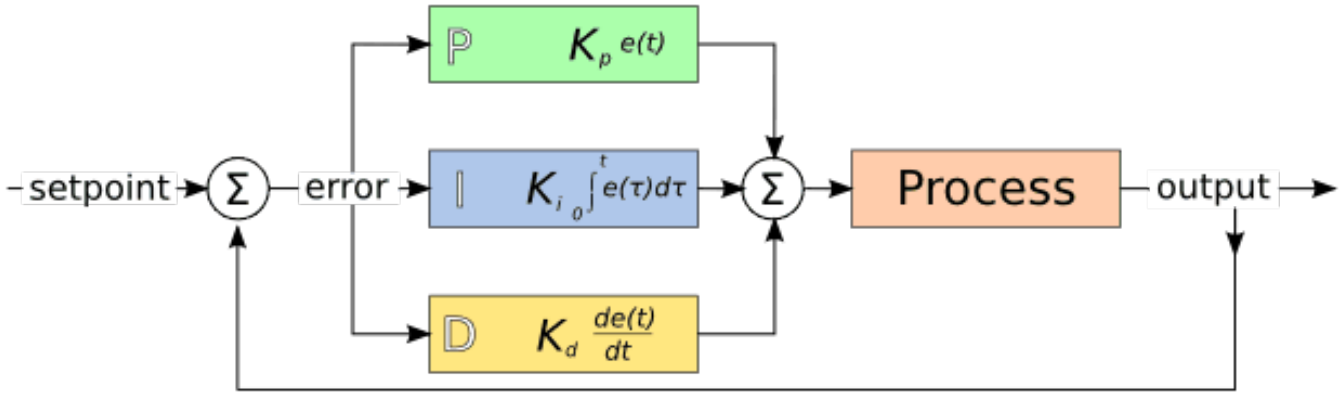


Figure 1.2: PID Loop

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.

1.2.2 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.

1.2.3 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).

1.2.4 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.

1.2.5 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.

1.2.6 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.

1.3 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.

1.3.1 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.

Part II

Configuration

Chapter 2

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 Hyperthreading 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 lrun 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
```

You should see something like this:

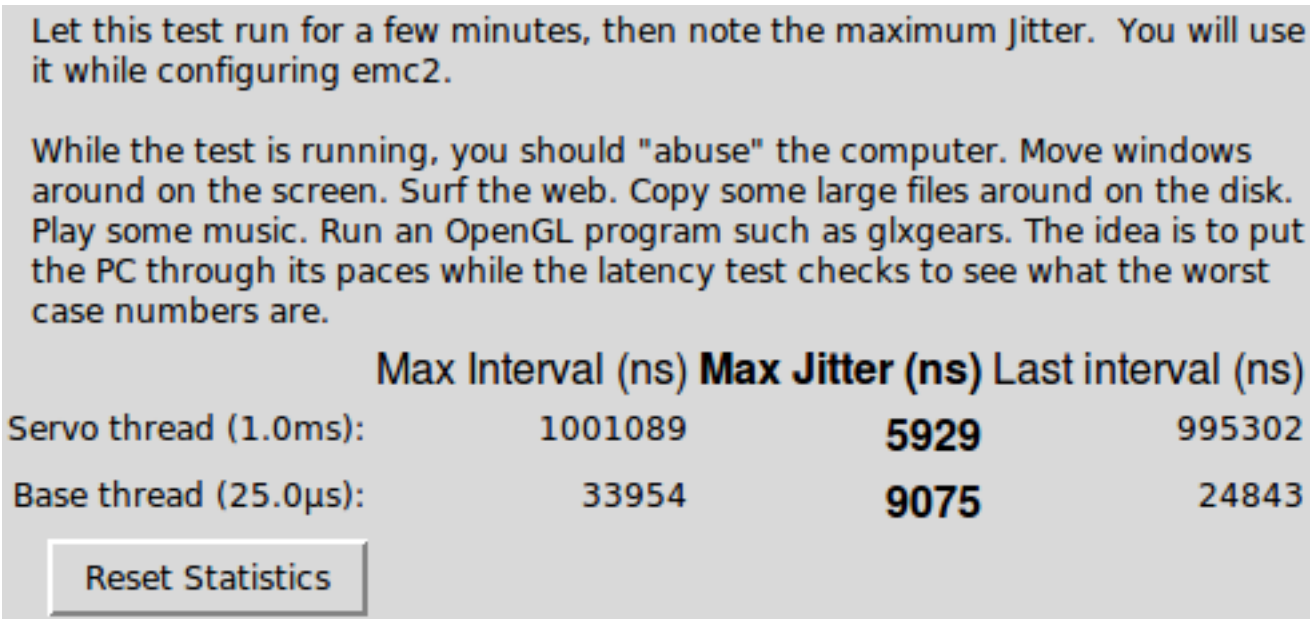


Figure 2.1: HAL Latency Test

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 numbers are the *max jitter*. In the example above, that is 9075 nanoseconds, or 9.075 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 that 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.

Note
LinuxCNC does not require bleeding edge hardware.

For more information on stepper tuning see the Stepper Tuning Chapter.

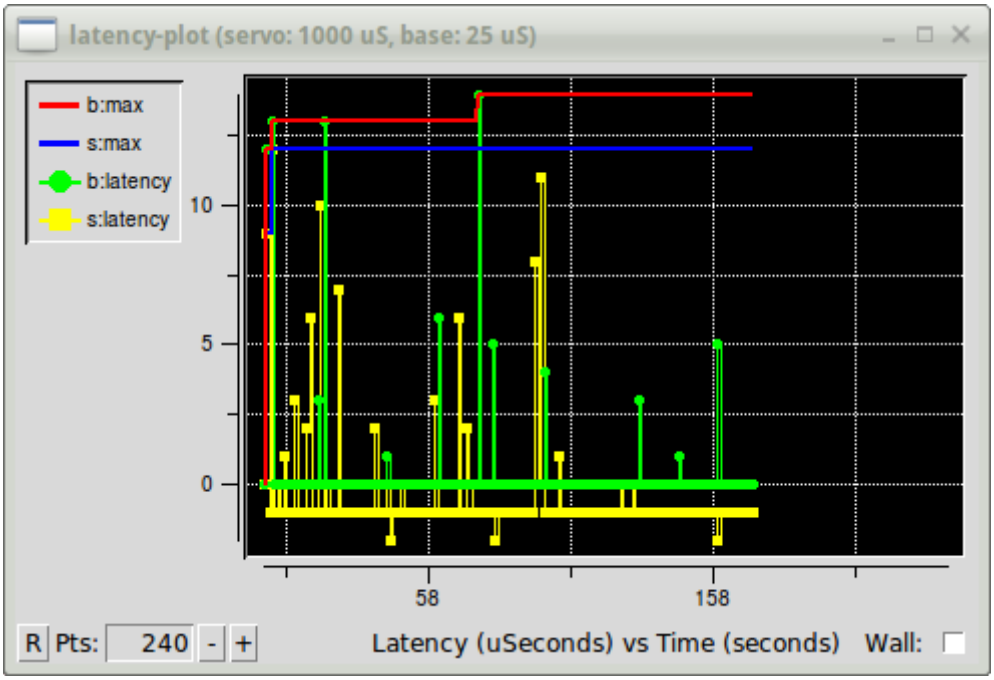
Additional command line tools are availalbe for examining latency when LinuxCNC is not running.

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 | -?      (this)
      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)
```



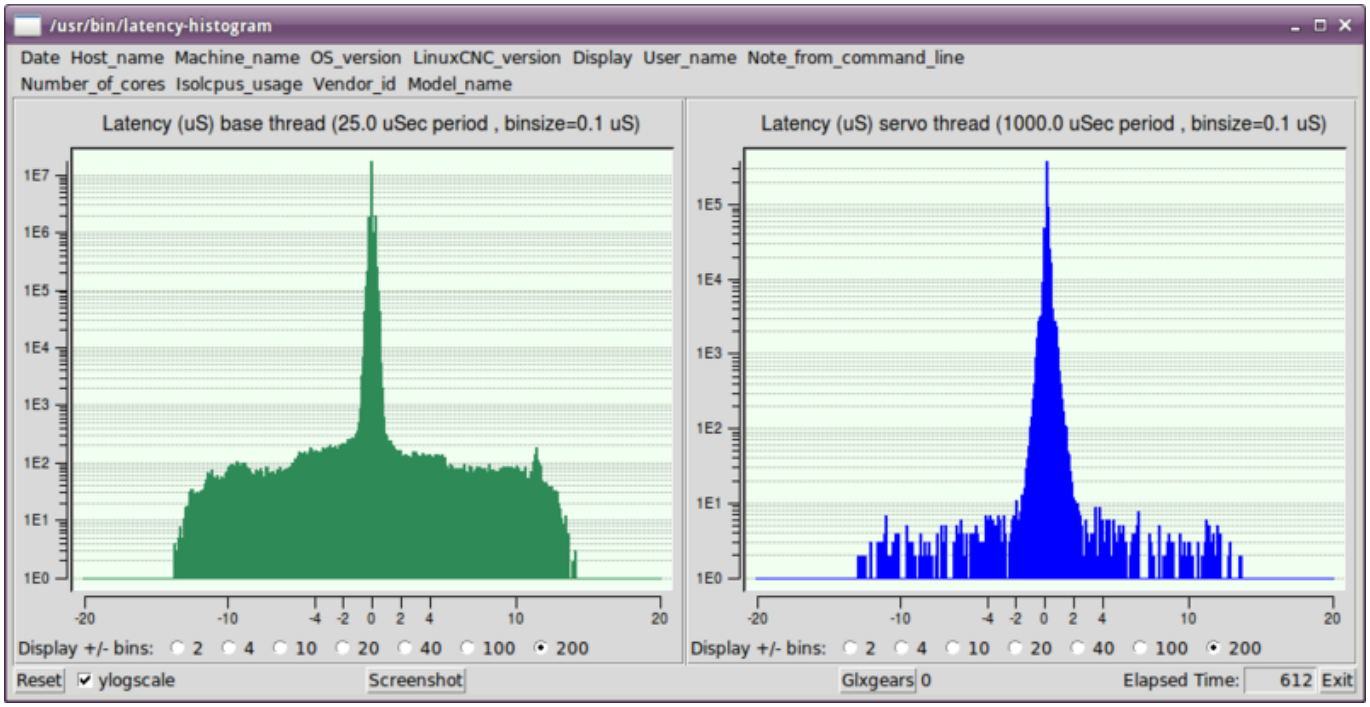
latency-histogram displays a histogram of latency (jitter) for a base and servo thread. Usage:

```
latency-histogram --help

Usage:
   latency-histogram --help | -?
or
   latency-histogram [Options]

Options:
  --version         (show version and exit)
  --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)
```

```
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
```



Chapter 3

Starting LinuxCNC

3.1 Running LinuxCNC

LinuxCNC is started with the script file *linuxcnc*.

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

LINUXCNC SCRIPT OPTIONS

- *-v* = verbose - prints info as it works
- *-d* = echoes script commands to screen for debugging

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.

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.

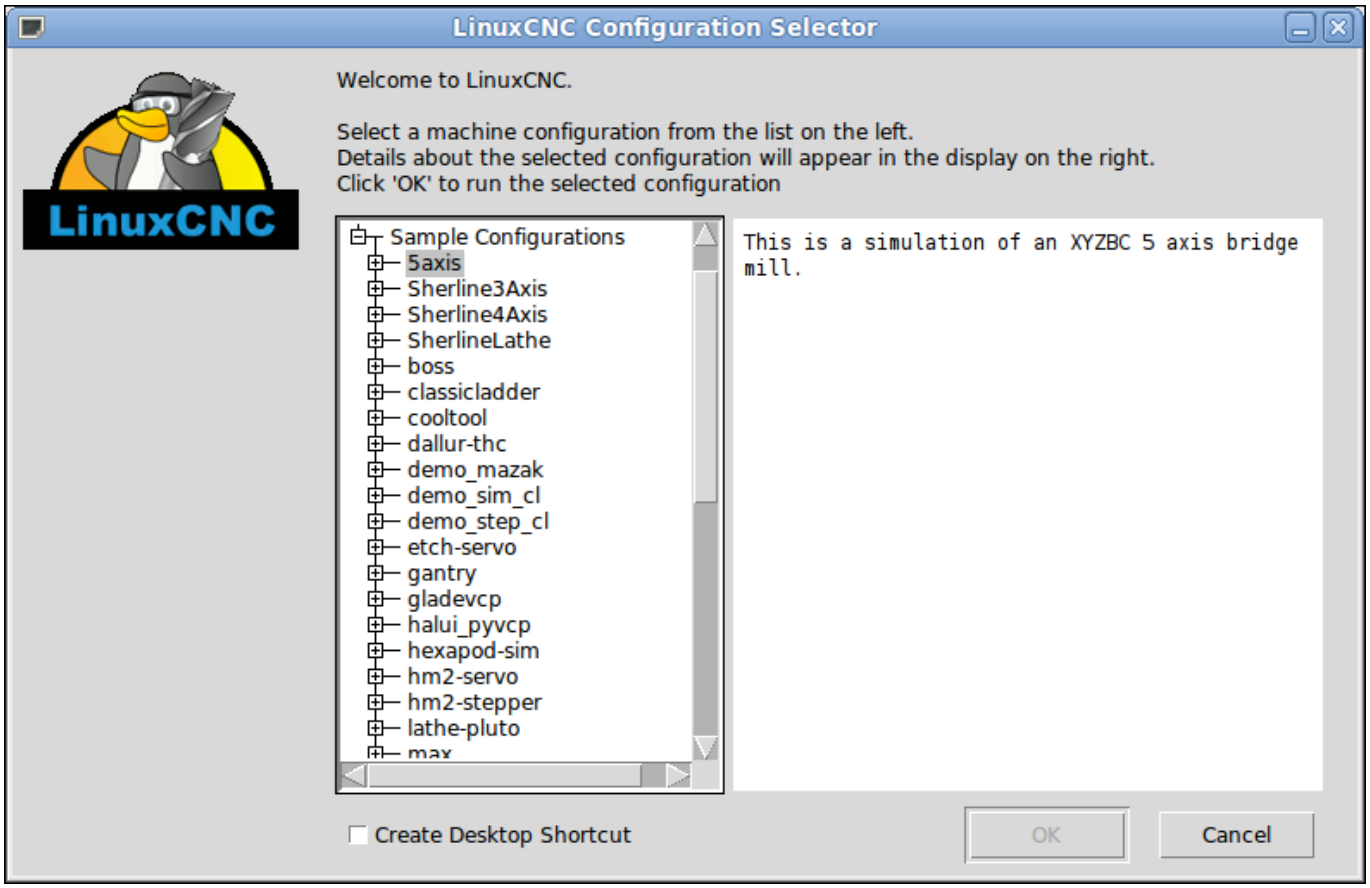


Figure 3.1: Configuration Selector

3.2 Files Used for Configuration

LinuxCNC is configured with human readable text files. All of these files can be read and edited in any of the common text file editors available with most any Linux distribution.[1] You'll need to be a bit careful when you edit these files. Some mistakes will cause the start up to fail. These files are read whenever the software starts up. Some of them are read repeatedly while the CNC is running.

Configuration files include

- *INI* The ini file overrides defaults that are compiled into the LinuxCNC code. It also provides sections that are read directly by the Hardware Abstraction Layer.
- *HAL* The HAL files start up process modules and provide linkages between LinuxCNC signals and specific hardware pins.
- *VAR* The var file is a way for the interpreter to save some values from one run to the next. These values are saved from one run to another but not always saved immediately. See the Parameters section of the G Code Manual for information on what each parameter is.
- *TBL* The tbl file saves tool information. See the User Manual Tool File section for more info.
- *NML* The nml file configures the communication channels used by the LinuxCNC. It is normally setup to run all of the communication within a single computer but can be modified to communicate between several computers.

[1] Don't confuse a text editor with a word processor. A text editor like gedit or kwrite produce files that are plain text. They also produce lines of text that are separated from each other. A word processor like Libre Office produces files with paragraphs and word wrapping and lots of embedded codes that control font size and such. A text editor does none of this.

- *linuxcncrc* This file saves user specific information and is created to save the name of the directory when the user first selects an LinuxCNC configuration.[2]

Items marked **(in HAL)** are used by the sample HAL files and are suggested as a good convention. Other items are used by LinuxCNC directly, and must always have the section and item names given.

3.3 TWOPASS

LinuxCNC 2.5 supports TWOPASS processing of hal configuration files that can help in the modularization and readability of hal files. (Hal files are specified in an LinuxCNC ini file in the HAL stanza as [HAL]HALFILE=filename).

Normally, a set of one or more hal 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, 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. No hal commands are executed in this initial pass.

After the initial pass, the modules are loaded 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.

[2] Usually this file is in the users home directory (e.g. /home/user/)

A second pass is then made to execute all of the other hal 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=xacel,yacel,zacel
```

results in components sensibly named xvel,yvel,zvel, xacel,yacel,zacel.

Many comps supplied with the distribution are created with the comp utility and support the names= option. These include the common logic components that are the glue of many hal configurations.

User-created comps that use the comp utility automatically support the names= option as well. In addition to comps generated with the comp utility, numerous other comps support the names=option. Comps that support names= option include: at_pid, encoder, encoder_ratio, pid, siggen, and sim_encoder.

Twopass processing occurs before the loading of a gui. When using a [HAL]POSTGUI_HALFILE, it is convenient to place all the loadrt statements for components needed in a halfile that is loaded earlier.

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
```

Examples of TWOPASS usage for a simulator are included in the directories:

```
configs/sim/axis/twopass/


configs/sim/axis/simtcl/
```

Chapter 4

INI Configuration

4.1 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.

4.1.1 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
```

4.1.2 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:

- *[EMC]* general information
- *[DISPLAY]* settings related to the graphical user interface
- *[FILTER]* settings input filter programs
- *[RS274NGC]* settings used by the g-code interpreter
- *[EMCMOT]* settings used by the real time motion controller
- *[TASK]* settings used by the task controller
- *[HAL]* specifies .hal files
- *[HALUI]* MDI commands used by HALUI
- *[APPLICATIONS]* Other applications to be started by LinuxCNC
- *[TRAJ]* additional settings used by the real time motion controller
- *[AXIS_n]* individual axis variables
- *[EMCIO]* settings used by the I/O Controller

4.1.3 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*.

4.1.4 Custom Sections and Variables

Most sample configurations use custom sections and variables to put all of the settings into one location for convenience.

To use a custom section variable in your HAL file add the section and variable to the INI file.

Custom Section Example

```
[OFFSETS]
OFFSET_1 = 0.1234
```

To add a custom variable to a LinuxCNC section simply include the variable in that section.

Custom Variable Example

```
[AXIS_0]
TYPE = LINEAR
...
SCALE = 16000
```

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 [OFFSETS]OFFSET_1
setp stepgen.0.position-scale [AXIS_0]SCALE
```

Note
The value stored in the variable must match the type specified by the component pin.

4.1.5 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 axis_0.inc
#INCLUDE ../parallel/axis_1.inc
#INCLUDE below/axis_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.

4.2 INI File Sections

4.2.1 [EMC] Section

- *VERSION = $Revision: 1.3 $* - The version number for the INI file. The value shown here looks odd because it is automatically updated when using the Revision Control System. It's a good idea to change this number each time you revise your file. If you want to edit this manually just change the number and leave the other tags alone.
- *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.

4.2.2 [DISPLAY] Section

Different user interface programs use different options, and not every option is supported by every user interface. The main two interfaces for LinuxCNC are AXIS and Touchy. There are several newer interfaces, like gmoccapy 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*, *keystick*, *mini*, *tklinuxcnc*, *xemc*,
- *POSITION_OFFSET = RELATIVE* - The coordinate system (RELATIVE or MACHINE) to show when the user interface starts. The RELATIVE coordinate system reflects the G92 and G5x coordinate offsets currently in effect.
- *POSITION_FEEDBACK = ACTUAL* - The coordinate value (COMMANDED or ACTUAL) to show when the user interface starts. The COMMANDED position is the ideal position requested by LinuxCNC. The ACTUAL position is the feedback position of the motors.
- *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 useful as it's dangerous to run a program with a too low spindle speed).
- *MAX_SPINDLE_OVERRIDE = 1.0* - The maximum spindle override the user may select. 1.0 means 100% of the programmed spindle speed.
- *DEFAULT_SPINDLE_SPEED = 100* - The default spindle RPM when the spindle is started in manual mode. This is not the minimum speed. In AXIS this defaults to 1 RPM if this setting is not present.
- *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 = 0.05* - Cycle time in seconds that display will sleep between polls.

Note
The following [DISPLAY] items are for the AXIS interface only, many of them are used also from gmoccapy, see the gmoccapy document for 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.
- *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.
- *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".
- *PYVCP = /filename.xml* - The PyVCP panel description file. See the PyVCP section for more information.
- *LATHE = 1* - This displays in lathe mode with a top view and with Radius and Diameter on the DRO.
- *GEOMETRY = XYZABCUVW* - Controls the preview and backplot of rotary motion. This item consists of a sequence of axis letters, optionally preceded by a "-" sign. Only axes defined in *[TRAJ]AXES* should be used. This sequence specifies the order in which the effect of each axis is applied, with a "-" inverting the sense of the rotation. The proper GEOMETRY string depends on the machine configuration and the kinematics used to control it. The example string GEOMETRY=XYZBCUVW is for a 5-axis machine where kinematics causes UVW to move in the coordinate system of the tool and XYZ to move in the coordinate system of the material. The order of the letters is important, because it expresses the order in which the different transformations are applied. For example rotating around C then B is different than rotating around B then C. Geometry has no effect without a rotary axis.
- *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%).[1]
- *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.

Note
The following [DISPLAY] item is used by the TKLinuxCNC interface only.

- *HELP_FILE = tklinucnc.txt* - Path to help file.

[1] In LinuxCNC 2.4 and earlier, the default value was 128.

4.2.3 [FILTER] Section

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 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-gcode converter included with LinuxCNC:

- *PROGRAM_EXTENSION = .png,.gif,.jpg Greyscale Depth Image*
  - *png = image-to-gcode*
  - *gif = image-to-gcode*
  - *jpg = 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. Many more g-code generators are on the LinuxCNC Wiki site http://wiki.linuxcnc.org/.

If the environment variable AXIS_PROGRESS_BAR is set, then lines written to stderr of the form

- *FILTER_PROGRESS=%d*

sets the AXIS progress bar to the given percentage. This feature should be used by any filter that runs for a long time.

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 python

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:])
```

4.2.4 [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:lathesubs:millsubs* - Specifies a colon (:) separated list of up to 10 directories to be searched when single-file subroutines are specified in gcode. 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.
- *USER_M_PATH = myfuncs:/tmp/mcodes:experimentalmcodes* - 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.
- *USER_DEFINED_FUNCTION_MAX_DIRS=5*. The maximum number of directories defined at compile time.

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.

4.2.5 [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.

4.2.6 [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 gcode. 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.

4.2.7 [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 using a system library of halfiles.
- *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 section on HALTCL 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.

  The TWOPASS item 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. Example:

```
TWOPASS = nodelete verbose
```

See the section on TWOPASS for more information.

Some GUIs support halfiles that are processed after the GUI is started in order to connect hal 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]
HALFILE = file_1.hal
...
HALFILE = file_n.hal
HALFILE = file_with_all_loads_for_postgui.hal
...
POSTGUI_HALFILE = the_postgui_file.hal
```

- *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 halfiles include Touchy, Axis, Gscreen, and Gmoccapy.

  ```
  See section <<sec:pyvcp-with-axis,pyVCP with Axis>> Section for more information.
  ```

- *HALUI = halui* - adds the HAL user interface pins. For more information see the HAL User Interface chapter.

4.2.8 [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.

4.2.9 [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 beore 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 previuosly saved watchlist. Since linuxcnc sets the working directory to the directory for the inifile, 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
    ```

4.2.10 [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 IPS^2, and a servo period of 0.001 sec 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 / ARC_BLEND_RAMP_FREQ, we don't bother with a trapezoidal velocity profile on that segment, and use constant acceleration. (Setting ARC_BLEND_RAMP_FREQ = 1000 is equivalent to always using trapezoidal acceleration, if the servo loop is 1kHz).

  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 20Hz 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 20Hz to 200Hz should be reasonable to start.

  Finally, no amount of tweaking will speed up a toolpath with lots of small, tight corners, since you're limited by cornering acceleration.

- *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. This has no effect on the mapping from G-code axis names (X- Y- Z-) to joint numbers—for *trivial kinematics*, X is always joint 0, A is always joint 3, and U is always joint 6, and so on. It is permitted to write an axis name twice (e.g., X Y Y Z for a gantry machine) but this has no effect.

- *AXES = 3* - One more than the number of the highest joint number in the system. For an XYZ machine, the joints are numbered 0, 1 and 2; in this case AXES should be 3. For an XYUV machine using *trivial kinematics*, the V joint is numbered 7 and therefore AXES should be 8. For a machine with nontrivial kinematics (e.g., scarakins) this will generally be the number of controlled joints.

- *JOINTS = 3* - (This config variable is used by the Axis GUI only, not by the trajectory planner in the motion controller.) Specifies the number of joints (motors) in the system. For example, an 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.

- *HOME = 0 0 0* - Coordinates of the homed position of each axis. Again for a fourth axis you will need 0 0 0 0. This value is only used for machines with nontrivial kinematics. On machines with trivial kinematics this value is ignored.

- *LINEAR_UNITS = <units>* - Specifies the *machine units* for linear axes. Possible choices are (in, inch, imperial, metric, mm). 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* (2pi 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_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_ACCELERATION = 2.0* - In machines with nontrivial kinematics, the acceleration used for "teleop" (Cartesian space) jogs, in *machine units* per second per second.
- *MAX_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_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.
- *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. Setting NO_FORCE_HOMING = 1 allows the user to make MDI moves and run programs without homing the machine first. Interfaces without homing ability will need to have this option set to 1.

Warning
Using this will allow the machine to go beyond the soft limits while in operation. It is not generally desirable to allow this.

4.2.11 [AXIS_<num>] Section

The [AXIS_0], [AXIS_1], etc. sections contains general parameters for the individual components in the axis control module. The axis section names begin numbering at 0, and run through the number of axes specified in the [TRAJ] AXES entry minus 1.

Typically (but not always):

- AXIS_0 = X
- AXIS_1 = Y
- AXIS_2 = Z
- AXIS_3 = A
- AXIS_4 = B
- AXIS_5 = C
- AXIS_6 = U
- AXIS_7 = V
- AXIS_8 = W
- *TYPE = LINEAR* - The type of axes, either LINEAR or ANGULAR.
- *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 = 1* - When this is set to 1 a G0 move for this axis will initiate an unlock with axis.N.unlock pin then wait for the axis.N.is-unlocked pin then move the axis at the rapid rate for that axis. After the move the axis.N.unlock will be false and motion will wait for axis.N.is-unlocked to go false. Moving with other axes is not allowed when moving a locked rotary axis.
- *UNITS = INCH* - If specified, this setting overrides the related [TRAJ] UNITS setting. (e.g., [TRAJ]LINEAR_UNITS if the TYPE of this axis is LINEAR, [TRAJ]ANGULAR_UNITS if the TYPE of this axis is ANGULAR)
- *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.
- *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 axis. If backlash is added to an axis and you are using steppers the STEPGEN_MAXACCEL must be increased to 1.5 to 2 times the MAX_ACCELERATION for the axis.
- *COMP_FILE = file.extension* - A file holding compensation structure for the axis. The file could be named xscrew.comp, for example, for the X axis. File names are case sensitive and can contain letters and/or numbers. The values are triplets per line separated by a space. The first value is nominal (where it should be). The second and third values depend on the setting of COMP_FILE_TYPE. Currently the limit inside LinuxCNC is for 256 triplets per axis. If COMP_FILE is specified, BACKLASH is ignored. Compensation file values are in machine units.
- *COMP_FILE_TYPE = 0 or 1* -
  - *If 0:* The second and third values specify the forward position (where the axis is while traveling forward) and the reverse position (where the axis is while traveling reverse), positions which correspond to the nominal position.'
  - *If 1:* The second and third values specify the forward trim (how far from nominal while traveling forward) and the reverse trim (how far from nominal while traveling in reverse), positions which correspond to the nominal position.

```
Example triplet with COMP_FILE_TYPE = 0: 1.00 1.01 0.99 +
Example triplet with COMP_FILE_TYPE = 1: 1.00 0.01 -0.01
```

- *MIN_LIMIT = -1000* - The minimum limit (soft limit) for axis motion, in machine units. When this limit is exceeded, the controller aborts axis motion.
- *MAX_LIMIT = 1000* - The maximum limit (soft limit) for axis motion, in machine units. When this limit is exceeded, the controller aborts axis motion.
- *MIN_FERROR = 0.010* - This is the value in machine units by which the axis 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. The following polarity values determine how inputs are interpreted and how outputs are applied. They can usually be set via trial-and-error since there are only two possibilities. The LinuxCNC Servo Axis Calibration utility program (in the AXIS interface menu Machine/Calibration and in TkLinuxCNC it is under Setting/Calibration) can be used to set these and more interactively and verify their results so that the proper values can be put in the INI file with a minimum of trouble.

4.2.11.1 Homing

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 axis 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 axis rapid velocity is used. Must be a positive number.
- *HOME_USE_INDEX = NO* - If the encoder used for this axis 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_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 axis 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 axis 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_SEQUENCE = <n>* - Used to define the "Home All" sequence. <n> starts at 0 and no numbers may be skipped. If left out or set to -1 the joint will not be homed by the "Home All" function. More than one axis can be homed at the same time.
- *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.

4.2.11.2 Servo

These parameters are relevant to axes 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:

$$\frac{1\,revolution}{1000\,lines} X \frac{1\,line}{4\,pulse/line} X \frac{0.2\,units}{1\,revolution} = \frac{0.200\,units}{4000\,pulses} = \frac{0.00005\,units}{1\,pulse}$$

- *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 axis servo. This value multiplies the error between commanded and actual position in machine units, resulting in a contribution to the computed voltage for the motor amplifier. The units on the P gain are volts per machine unit, e.g., $\frac{volts}{unit}$
- *I = 0* - The integral gain for the axis servo. The value multiplies the cumulative error between commanded and actual position in machine units, resulting in a contribution to the computed voltage for the motor amplifier. The units on the I gain are volts per machine unit second, e.g., $\frac{volts}{unit\,second}$
- *D = 0* - The derivative gain for the axis 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., $\frac{volts}{unit\,second}$
- *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., $\frac{volts}{unit}$
- *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., $\frac{volts}{unit\,second}$
- *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., $\frac{volts}{unit\,second^2}$
- *OUTPUT_SCALE = 1.000* -
- *OUTPUT_OFFSET = 0.000* - 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: $raw = \frac{output - offset}{scale}$

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 volt results in 250 mm/sec velocity.

$$amplifier[volts]=(output[\frac{mm}{sec}]-offset[\frac{mm}{sec}])/250\,\frac{mm}{secvolt}$$

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 4.1: 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 axis one machine unit will be equal to the setting of LINEAR_UNITS. For an angular axis 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:

$$input\,scale=2000\,\frac{counts}{rev}*10\,\frac{rev}{inch}=20000\,\frac{counts}{inch}$$

4.2.11.3 Stepper

These parameters are relevant to axes 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 axis one machine unit will be equal to the setting of LINEAR_UNITS. For an angular axis 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:

$$input\,scale = \frac{2\ steps}{1.8\ degrees} * 360\ \frac{degree}{rev} * 10\ \frac{rev}{inch} = 4000\ \frac{steps}{inch}$$

- *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 axis one machine unit will be equal to the setting of LINEAR_UNITS. For an angular axis 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:

$$input\,scale = 2000\ \frac{counts}{rev} * 10\ \frac{rev}{inch} = 20000\ \frac{counts}{inch}$$

- *STEPGEN_MAXACCEL = 21.0* - Acceleration limit for the step generator. This should be 1% to 10% larger than the axis MAX_ACCELERATION. This value improves the tuning of stepgen's "position loop". If you have added backlash compensation to an axis 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 axis MAX_VELOCITY. Subsequent testing has shown that use of STEPGEN_MAXVEL does not improve the tuning of stepgen's position loop.

4.2.12 [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.
- *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 on G30 and Parameters see the G Code Manual.
- *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.

Chapter 5

Homing Configuration

5.1 Overview

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.

5.2 Homing Sequence

There are four possible homing sequences defined by the sign of SEARCH_VEL and LATCH_VEL, along with the associated configuration parameters as shown in the following table. Two basic conditions exist, SEARCH_VEL and LATCH_VEL are the same sign or they are opposite signs. For a more detailed description of what each configuration parameter does, see the following section.

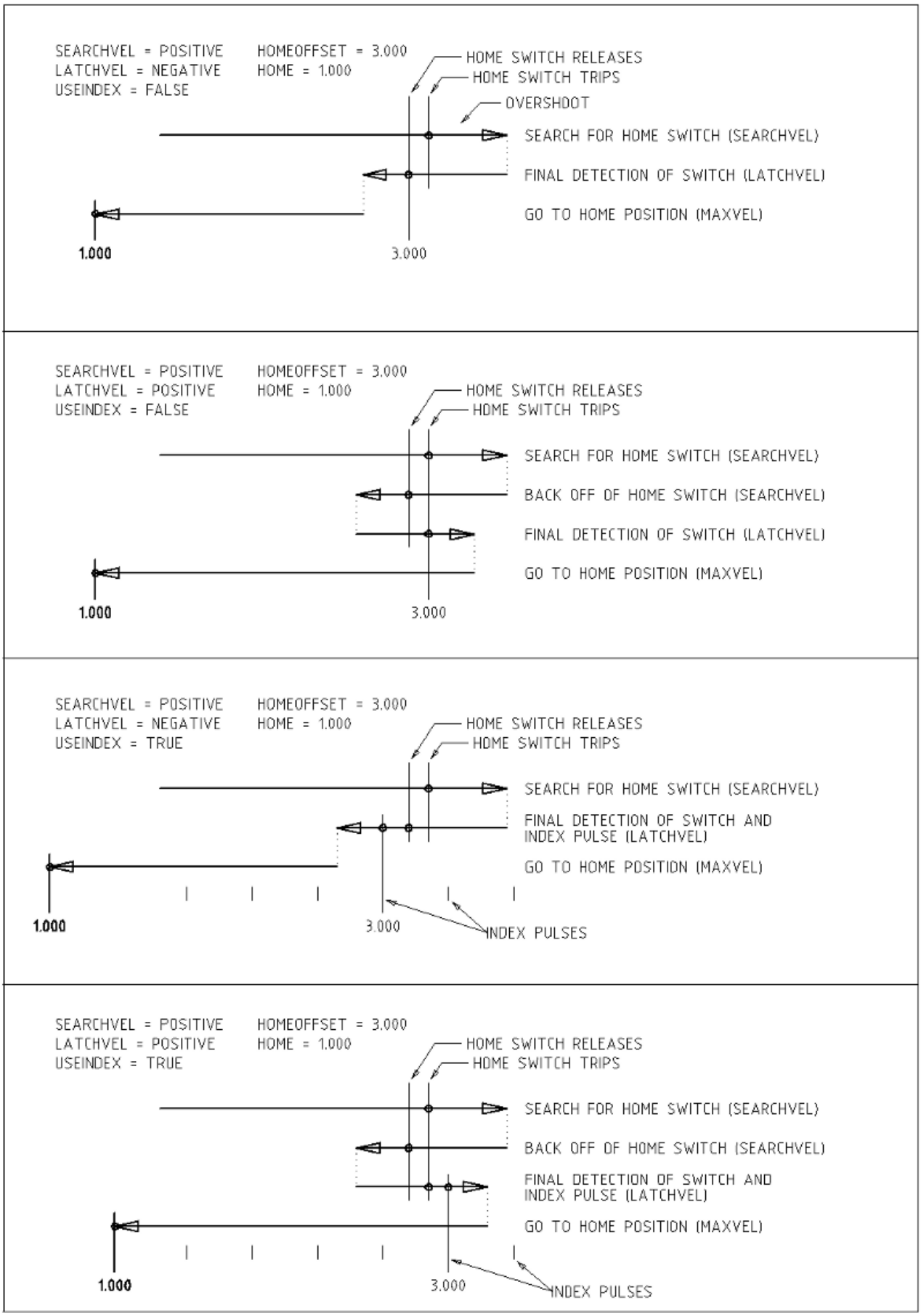


Figure 5.1: Homing Sequences

5.3 Configuration

The following determines exactly how the home sequence behaves. They are defined in an [AXIS] section of the inifile.

Homing Type	SEARCH_VEL	LATCH_VEL	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.

5.3.1 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.

5.3.2 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.

5.3.3 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.

5.3.4 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 axis while homing. Setting this to YES will not ignore limit inputs for other axes. If you do not have

a separate home switch set this to YES and case connect the limit switch signal to the home switch input in HAL. LinuxCNC will ignore the limit switch input for this axis while homing. To use only one input for all homing and limits you will have to block the limit signals of the axes not homing in HAL and home one axis at a time.

5.3.5 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.

5.3.6 HOME_OFFSET

Contains the location of the home switch or index pulse, in joint coordinates. It can also be treated as the distance between the point where the switch or index pulse is latched and the zero point of the joint. After detecting the index pulse, LinuxCNC sets the joint coordinate of the current point to HOME_OFFSET. The default value is zero.

5.3.7 HOME

The position that the joint will go to upon completion of the homing sequence. After detecting the index pulse, 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. Since the joint is now homed, there should be no risk of crashing the machine, and a rapid move is the quickest way to finish the homing sequence. [1]

5.3.8 HOME_IS_SHARED

If there is not a separate home switch input for this axis, 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.

5.3.9 HOME_SEQUENCE

Used to define a multi-axis homing sequence HOME ALL and enforce homing order (e.g., Z may not be homed if X is not yet homed). An axis may be homed after all axes with a lower HOME_SEQUENCE have already been homed and are at the HOME_OFFSET. If two axes have the same HOME_SEQUENCE, they may be homed at the same time. If HOME_SEQUENCE is -1 or not specified then this joint will not be homed by the HOME ALL sequence. HOME_SEQUENCE numbers start with 0 and there may be no unused numbers.

5.3.10 VOLATILE_HOME

If this setting is true, this axis becomes unhomed whenever the machine transitions into the OFF state. This is appropriate for any axis that does not maintain position when the axis drive is off. Some stepper drives, especially microstep drives, may need this.

5.3.11 LOCKING_INDEXER

If this axis is a locking rotary indexer, it will unlock before homing, and lock afterward.

[1] The distinction between *home_offset* and *home* is that *home_offset* first establishes the scale location 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.

5.3.12 Immediate Homing

If an axis does not have home switches or does not have a logical home position like a rotary axis and you want that axis to home at the current position when the "Home All" button is pressed in Axis the following ini entries for that axis are needed.

1. SEARCH_VEL = 0
2. LATCH_VEL = 0
3. USE_INDEX = NO
4. HOME_SEQUENCE = 0

Chapter 6

Lathe Configuration

6.1 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.

6.2 INI Settings

The following .ini settings are needed for lathe mode in Axis in addition to or replacing normal settings in the .ini file. Gmoccapy uses also the mentioned settings, but does offer additional settings, check the gmoccapy Section for details.

```
[DISPLAY]
DISPLAY = axis
LATHE = 1

[TRAJ]
AXES = 3
COORDINATES = X Z
[AXIS_0]
...
[AXIS_2]
...
```

Chapter 7

HALTCL Files

The halcmd language excels in specifiying 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 halfiles.

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.

7.1 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.

7.2 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
```

7.3 Haltcl Inifile variables

Inifile 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 substition 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:

```
[AXIS_0]
MAX_VELOCITY = 4
```

is expressed as [AXIS_0]MAX_VELOCITY in .hal files for halcmd and as $::AXIS_0(MAX_VELOCITY) in .tcl files for haltcl

7.4 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
```

7.5 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 AXIS_0 radius
setp scale.0.gain 6.28/360.0*$::AXIS_0(radius)*60.0
```

Whitespace in the bare expression is not allowed, use quotes for that:

```
setp scale.0.gain "6.28 / 360.0 * $::AXIS_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)}]
```

7.6 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 inifiles to have a manually calculated value.

```
[AXIS_0]
MAXACCEL = 10.0
STEPGEN_MAXACCEL = 10.5
```

With haltcl, you can use tcl commands to do the computation and eliminate the STEPGEN_MAXACCEL inifile item altogether.

```
setp stepgen.0.maxaccel $::AXIS_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
foreach axis {X Y Z A B C U V W} axno {0 1 2 3 4 5 6 7 8} {
  # 'list pin' returns an empty list if the pin doesn't exist
  if {[hal list pin axis.$axno.motor-pos-cmd] == {}} {
    continue
  }
  net ${axis}pos axis.$axno.motor-pos-cmd => axis.$axno.motor-pos-fb \
                                         => ddt.$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]
```

7.7 Haltcl Interactive

The halrun command recognizes haltcl files. With the -T option, haltcl can be run interaactively as a tcl interpreter. This capability is useful for testing and for standalone hal applications.

Example

```
$ halrun -T haltclfile.tcl
```

7.8 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.

Chapter 8

Core Components

See also the man pages *motion(9)*.

8.1 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. The default is 4 each.

Pin names starting with *axis* are actually joint values, but the pins and parameters are still called *axis.N*. They 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])]
```

- *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*.

8.1.1 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.

8.1.2 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*.
- *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.on-soft-limit* - (bit, out) TRUE when the machine is on a soft limit.
- *motion.probe-input* - (bit, in) *G38.x* 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 from the F=n setting in the G Code file. No feed overrides or any other adjustments are applied to this pin.
- *motion.spindle-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.
- *motion.spindle-brake* - (bit, out) TRUE when the spindle brake should be applied.

- *motion.spindle-forward* - (bit, out) TRUE when the spindle should rotate forward.
- *motion.spindle-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.
- *motion.spindle-inhibit* - (bit, in) When this bit is TRUE, the spindle speed is set to 0.
- *motion.spindle-on* - (bit, out) TRUE when spindle should rotate.
- *motion.spindle-reverse* - (bit, out) TRUE when the spindle should rotate backward
- *motion.spindle-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.
- *motion.spindle-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 *motion.spindle-speed-out-rps*.
- *motion.spindle-speed-out* - (float, out) Commanded spindle speed in rotations per minute. Positive for spindle forward (*M3*), negative for spindle reverse (*M4*).
- *motion.spindle-speed-out-abs* - (float, out) Commanded spindle speed in rotations per minute. This will always be a positive number.
- *motion.spindle-speed-out-rps* - (float, out) Commanded spindle speed in rotations per second. Positive for spindle forward (*M3*), negative for spindle reverse (*M4*).
- *motion.spindle-speed-out-rps-abs* - (float, out) Commanded spindle speed in rotations per second. This will always be a positive number.
- *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.spindle-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.
- `motion.spindle-orient-mode` - (s32,out) Desired spindle rotation mode M19. Default 0.
- `motion.spindle-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.
- `motion.spindle-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.
- `motion.spindle-orient-fault` - (s32, in) Fault code input for orient cycle. Any value other than zero will cause the orient cycle to abort.
- `motion.spindle-lock` - (bit, out) Spindle orient complete pin. Cleared by any of M3,M4,M5.

8.1.2.1 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
- orienation complete mode.

When an M19 is executed, the spindle changes to *searching for desired orientation* , and the `spindle-orient` HAL pin is asserted. The desired target position is specified by the `spindle-orient-angle` and `spindle-orient-fwd` pins and driven by the M19 R and P parameters.

The HAL support logic is expected to react to `spindle-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-is-oriented` pin.

Motion then acknowledges this by deasserting the `spindle-orient` pin and asserts the `spindle-locked` pin to indicate *orientation complete* mode. It also raises the `spindle-brake` pin. The spindle now is in *orientation complete* mode.

If, during `spindle-orient` being true, and `spindle-is-oriented` not yet asserted the `spindle-orient-fault` 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 positon, `spindle-orient-angle` and `spindle-orient-mode`.

8.1.3 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)
- *motion.servo.overruns* - (u32, RW) By noting large differences between successive values of *motion.servo.last-period* , the motion controller can determine that there has probably been a failure to meet its timing constraints. Each time such a failure is detected, this value is incremented.

8.1.4 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

8.2 Axis (Joints)

These pins and parameters are created by the realtime *motmod* module. These are actually joint values, but the pins and parameters are still called *axis.N*.[1] They are read and updated by the *motion-controller* function.

8.2.1 Pins

- *axis.N.active* - (bit, out)
- *axis.N.amp-enable-out* - (bit, out) TRUE if the amplifier for this joint should be enabled
- *axis.N.amp-fault-in* - (bit, in) Should be driven TRUE if an external fault is detected with the amplifier for this joint
- *axis.N.backlash-corr* - (float, out)
- *axis.N.backlash-filt* - (float, out)
- *axis.N.backlash-vel* - (float, out)
- *axis.N.coarse-pos-cmd* - (float, out)
- *axis.N.error* - (bit, out)
- *axis.N.f-error* - (float, out)
- *axis.N.f-error-lim* - (float, out)
- *axis.N.f-errored* - (bit, out)
- *axis.N.faulted* - (bit, out)
- *axis.N.free-pos-cmd* - (float, out)
- *axis.N.free-tp-enable* - (bit, out)
- *axis.N.free-vel-lim* - (float, out)
- *axis.N.home-sw-in* - (bit, in) Should be driven TRUE if the home switch for this joint is closed.
- *axis.N.homed* - (bit, out)
- *axis.N.homing* - (bit, out) TRUE if the joint is currently homing
- *axis.N.in-position* - (bit, out)
- *axis.N.index-enable* - (bit, I/O)
- *axis.N.jog-counts* - (s32, in) Connect to the *counts* pin of an external encoder to use a physical jog wheel.
- *axis.N.jog-enable* - (bit, in) When TRUE (and in manual mode), any change in *jog-counts* will result in motion. When false, *jog-counts* is ignored.
- *axis.N.jog-scale* - (float, in) Sets the distance moved for each count on *jog-counts*, in machine units.

[1] In *trivial kinematics* machines, there is a one-to-one correspondence between joints and axes.

- *axis.N.jog-vel-mode* - (bit, in) When FALSE (the default), the jogwheel operates in position mode. The axis will move exactly jog-scale units for each count, regardless of how long that might take. When TRUE, the wheel operates in velocity mode - motion stops when the wheel stops, even if that means the commanded motion is not completed.
- *axis.N.joint-pos-cmd* - (float, out) The joint (as opposed to motor) commanded position. There may be an offset between the joint and motor positions—for example, the homing process sets this offset.
- *axis.N.joint-pos-fb* - (float, out) The joint (as opposed to motor) feedback position.
- *axis.N.joint-vel-cmd* - (float, out)
- *axis.N.kb-jog-active* - (bit, out)
- *axis.N.motor-pos-cmd* - (float, out) The commanded position for this joint.
- *axis.N.motor-pos-fb* - (float, in) The actual position for this joint.
- *axis.N.neg-hard-limit* - (bit, out)
- *axis.N.pos-lim-sw-in* - (bit, in) Should be driven TRUE if the positive limit switch for this joint is closed.
- *axis.N.pos-hard-limit* - (bit, out)
- *axis.N.neg-lim-sw-in* - (bit, in) Should be driven TRUE if the negative limit switch for this joint is closed.
- *axis.N.wheel-jog-active* - (bit, out)

8.2.2 Parameters

- *axis.N.home-state* - Reflects the step of homing currently taking place.

8.3 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.

8.3.1 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-prep-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.

8.4 ini settings

A number of ini settings are made available as hal input pins.

8.4.1 Pins

- *ini.n.min_limit* - (float, in) [AXIS_n]MIN_LIMIT
- *ini.n.max_limit* - (float, in) [AXIS_n]MAX_LIMIT
- *ini.n.ferror* - (float, in) [AXIS_n]FERROR
- *ini.n.min_ferror* - (float, in) [AXIS_n]MIN_FERROR
- *ini.n.max_velocity* - (float, in) [AXIS_n]MAX_VELOCITY
- *ini.n.max_acceleration* - (float, in) [AXIS_n]MAX_ACCELERATION
- *ini.n.backlash* - (float, in) [AXIS_n]BACKLASH

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

Chapter 9

Stepper Configuration

9.1 Introduction

The preferred way to set up a standard stepper machine is with the Step Configuration Wizard. See the Getting Started Guide.

This chapter describes some of the more common settings for manually setting up a stepper based system. Because of the various possibilities of configuring LinuxCNC, it is very hard to document them all, and keep this document relatively short.

The most common LinuxCNC usage is for stepper based systems. 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 the sample config released along with LinuxCNC. The config is called stepper, and usually it is found in */etc/emc2/sample-configs/stepper*.

9.2 Maximum step rate

With software step generation, the maximum step rate is one step per two BASE_PERIODs 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.)

9.3 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 manual for more detailed 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 xylotex_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.

9.3.1 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 motion.spindle-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 => axis.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.  See the
### user manual!
###

# net homeswitches <= parport.0.pin-10-in
# net homeswitches => axis.0.home-sw-in
# net homeswitches => axis.1.home-sw-in
# net homeswitches => axis.2.home-sw-in

###
### Sample separate limit switches on the X axis (axis 0)
###

# net X-neg-limit parport.0.pin-11-in => axis.0.neg-lim-sw-in
# net X-pos-limit parport.0.pin-12-in => axis.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.
###

# net Xlimits parport.0.pin-13-in => axis.0.neg-lim-sw-in axis.0.pos-lim-sw-in
```

The lines starting with # are comments, and their only purpose is to guide the reader through the file.

9.3.2 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 section of the HAL Manual 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

9.3.3 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:

[1] the fastest thread in the LinuxCNC setup, usually the code gets executed every few tens of microseconds

```
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.

9.3.4 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
```

9.3.5 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 motion.spindle-speed-out => pwmgen.0.value
net spindle-on motion.spindle-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.

9.3.6 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.

9.3.7 External ESTOP button

As you can see in the standard_pinout.hal file by default the stepper configuration assumes no external ESTOP button. [2]

To add a simple external button you need to replace the line:

[2] An extensive explanation of hooking up ESTOP circuitry is explained in the wiki.linuxcnc.org and elsewhere in the Integrator Manual

```
net estop-loop iocontrol.0.user-enable-out iocontrol.0.emc-enable-in
```

with

```
net estop-loop parport.0.pin-01-in iocontrol.0.emc-enable-in
```

This assumes an ESTOP switch connected to pin 01 on the parport. As long as the switch will stay pushed[3], LinuxCNC will be in the ESTOP state. When the external button gets released LinuxCNC will immediately switch to the ESTOP-RESET state, and all you need to do is switch to Machine On and you'll be able to continue your work with LinuxCNC.

[3] make sure you use a maintained switch for ESTOP.

Chapter 10

Basic HAL Reference

10.1 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.

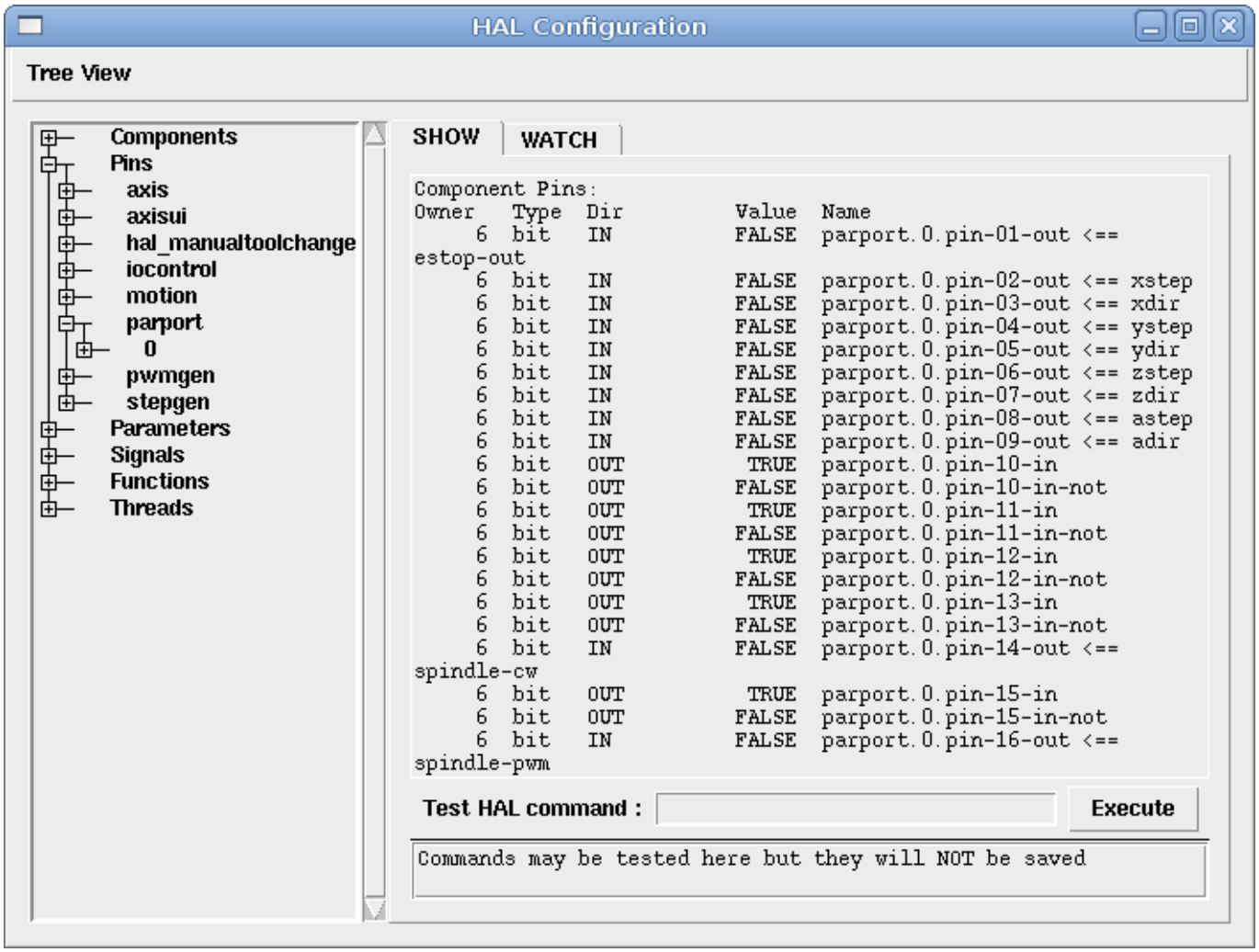


Figure 10.1: HAL Configuration Window

10.1.1 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.

The syntax and an example:

```
loadrt <component> <options>

loadrt mux4 count=1
```

10.1.2 addf

The command *addf* adds a real time component function to a thread. 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.

If you used the Stepper Config Wizard to generate your config you will have two threads.

- 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.

- 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.

The syntax and an example:

```
addf <component> <thread>

addf mux4 servo-thread
```

10.1.3 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.

The 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*.

10.1.4 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.

Syntax and Example:

```
net signal-name pin-name <optional arrow> <optional second pin-name>

net home-x axis.0.home-sw-in <= parport.0.pin-11-in
```

In the above example *home-x* is the signal name, *axis.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.

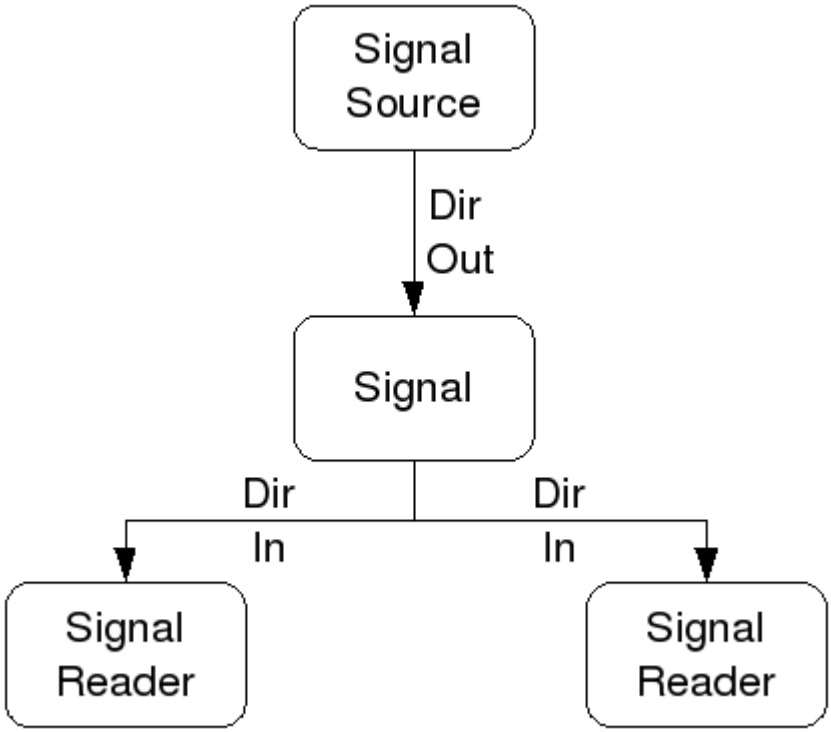


Figure 10.2: Signal Direction

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.

```
net xStep => parport.0.pin-02-out
```

I/O pins An I/O pin like encoder.N.index-enable can be read or set as allowed by the component.

10.1.5 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.

The syntax and an example:

```
setp <pin/parameter-name> <value>

setp parport.0.pin-08-out TRUE
```

10.1.6 sets

The command *sets* sets the value of a signal.

The syntax and an 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

10.1.7 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.

The syntax and an example:

```
unlinkp <pin-name>

unlinkp parport.0.pin-02-out
```

10.1.8 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.

10.1.8.1 linksp

The command *linksp* creates a *connection* between a signal and one pin.

The syntax and an example:

```
linksp <signal-name> <pin-name>
linksp X-step parport.0.pin-02-out
```

The *linksp* command has been superseded by the *net* command.

10.1.8.2 linkps

The command *linkps* creates a *connection* between one pin and one signal. It is the same as linksp but the arguments are reversed.

The syntax and an example:

```
linkps <pin-name> <signal-name>

linkps parport.0.pin-02-out X-Step
```

The *linkps* command has been superseded by the *net* command.

10.1.8.3 newsig

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.

The syntax and an example:

```
newsig <signame> <type>

newsig Xstep bit
```

More information can be found in the HAL manual or the man pages for halrun.

10.2 HAL Data

10.2.1 Bit

A bit value is an on or off.

- bit values = true or 1 and false or 0 (True, TRUE, true are all valid)

10.2.2 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 1000 bits of dynamic range.

For more information on floating point numbers see:

http://en.wikipedia.org/wiki/Floating_point

10.2.3 s32

An *s32* number is a whole number that can have a negative or positive value.

- s32 values = integer numbers -2147483648 to 2147483647

10.2.4 u32

A *u32* number is a whole number that is positive only.

- u32 values = integer numbers 0 to 4294967295

10.3 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.

10.4 HAL Components

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`

`.tmax`

Time is the number of CPU cycles it took to execute the function.

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.

10.5 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.

10.5.1 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)
```

Truth Table

in0	in1	out
False	False	False
True	False	False
False	True	False
True	True	True

10.5.2 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)
```

Truth Table

in	out
True	False
False	True

10.5.3 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)
```

Truth Table

in0	in1	out
True	False	True
True	True	True
False	True	True
False	False	False

10.5.4 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)
```

Truth Table

in0	in1	out
True	False	True
True	True	False
False	True	True
False	False	False

10.5.5 Logic Examples

An *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.

10.6 Conversion Components

10.6.1 weighted_sum

The weighted_sum converts a group of bits to an integer. The conversion is the sum of the *weights* of the bits that are on plus any offset. The weight of the m-th bit is 2^m. This is similar to a binary coded decimal but with more options. The *hold* bit stops processing the input changes so the *sum* will not change.

The following syntax is used to load the weighted_sum component.

```
loadrt weighted_sum wsum_sizes=size[,size,...]
```

Creates weighted sum groups each with the given number of input bits (size).

To update the weighted_sum you need to attach process_wsums to a thread.

```
addf process_wsums servo-thread
```

This updates the weighted_sum component.

In the following example clipped from the HAL Configuration window in Axis the bits *0* and *2* are true and there is no offset. The *weight* of 0 is 1 and the *weight* of 2 is 4 so the sum is 5.

weighted_sum

```
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
```

Chapter 11

Extending LinuxCNC

11.1 Introduction: Extending the RS274NGC Interpreter by Remapping Codes

11.1.1 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.

11.1.2 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 your 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 a an O-word procedure call.

In it's 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.

11.1.2.1 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.

11.1.2.2 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.

11.1.2.3 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 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 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.

11.2 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.

11.2.1 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.

11.2.2 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.

11.2.3 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.

11.2.4 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.

11.2.5 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. The 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 exists in the `[DISPLAY]NC_FILES` or `[RS274NGC]SUBROUTINE_PATH` directory.

A detailed discussion of REMAP parameters is found in the reference section below.

11.3 Configuring Remapping

11.3.1 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.

11.3.2 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.

11.3.3 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 by 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.

ABCDEFGHIJKMPQRSTUVWXYZ
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.

abcdefghijkmpqrstuvwxyz
: 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.

11.3.3.1 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`

11.3.3.2 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`

NB: 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.

11.3.3.3 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.

11.3.3.4 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.

11.4 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-directory>`
- 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
```

11.5 Remapping tool change-related codes: T, M6, M61

11.5.1 Overview

If you are unfamiliar with LinuxCNC internals, first read the How tool change currently works section (dire but necessary).

Note than 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.

11.5.2 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_POCKET() 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 cant handle will be done in Python prolog and epilog functions.

11.5.3 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 pockets `current_pocket` and `selected_pocket`:

```
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):

```
O<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>
;
O<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 cant be done in NGC code.

11.5.4 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.

11.5.5 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.

11.5.6 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.

11.5.7 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 epilog:

```
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_POCKET(self.selected_pocket, 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.

11.5.8 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.

```
[RS274NGC]
ON_ABORT_COMMAND=O <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)

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). `on_abort` receives a single parameter indicating the cause for calling the abort procedure, which might be used for conditional cleanup.

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.

11.5.9 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/abort-hot-comment/test.ngc`):

```
o100 if [..] (some error condition)
     (abort, Bad Things! p42=#42 q=#<q> ini=#<_ini[a]x> pin=#<_hal[component.pin])
o100 endif
```

NB: ini and HAL variable expansion need explicit enabling with FEATURE.

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).

11.6 Remapping other existing codes: S, M0, M1, M60

11.6.1 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.

11.6.2 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.

11.7 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.

11.8 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.

11.8.1 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.

11.8.2 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
```

11.9 Programming Embedded Python in the RS274NGC Interpreter

11.9.1 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 funtcions int this module extend or redefine the namespace of predefined named parameters, see adding predefined parameters.

task
: Task-related callables are expected here.

11.9.2 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.

11.9.3 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*).

11.9.4 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.

11.9.4.1 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
```

11.9.4.2 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 `endsub` with an explicit value. 1.0 if true. Set to 0.0 on each `call`. Exposed in Interp was `self.value_returned` (int).

See also `tests/interp/value-returned` for an example.

11.9.4.3 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)`.

.

11.9.4.4 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 th en signal the interpreter to continue. At this point the function is resumed at the statement following the `yield ..` statement.

11.9.4.5 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.

11.9.5 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.

11.9.5.1 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.

11.9.5.2 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.

11.9.5.3 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 occured

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
```

11.9.5.4 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/emccanon.cc`. The implementation of the Python functions can be found in `src/emc/rs274ncg/canonmodule.cc`.

11.9.6 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/introspect` 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.

11.10 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 aribtrary 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.

Ònly 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 isnt 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`.

11.11 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:

11.11.1 T: `prepare_prolog` and `prepare_epilog`

These wrap a NGC procedure for Tx Tool Prepare.

11.11.1.1 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.

11.11.1.2 Actions of `prepare_epilog`

- The NGC procedure is expected to return a positive value, otherwise and 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 be 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_POCKET(#<pocket>,#<tool>)` is executed.

11.11.2 M6: `change_prolog` and `change_epilog`

These wrap a NGC procedure for M6 Tool Change.

11.11.2.1 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 pocket number

11.11.2.2 Actions of `change_epilog`

- The NGC procedure is expected to return a positive value, otherwise and 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 be 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_TOOL(#<selected_pocket>)` is executed.
- The new tool parameters (offsets, diameter etc) are set.

11.11.3 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 paramters 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 give in the block, see below.

11.11.3.1 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.

11.11.3.2 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.

11.11.4 S (Set Speed) : `setspeed_prolog` and `setspeed_epilog`

TBD

11.11.5 F (Set Feed) : `setfeed_prolog` and `setfeed_epilog`

TBD

11.11.6 M61 Set tool number : `settool_prolog` and `settool_epilog`

TBD

11.12 Remapped code execution

11.12.1 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.

11.12.2 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.

11.12.3 Sequence number during remaps

Sequence numbers are propagated and restored like with O-word calls. See `tests/remap/nested-remaps/word` for the regression test, which shows sequence number tracking during nested remaps three levels deep.

11.12.4 Debugging flags

The following flags are relevant for remapping and Python - related execution:

```
EMC_DEBUG_OWORD             0x00002000  traces execution of O-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*.

11.12.5 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 PydDev 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 the `pydevd` variable in `util.py` to reflect this directory location.
- `import pydevd` in your Python module - see example `util.py` and `remap.py`

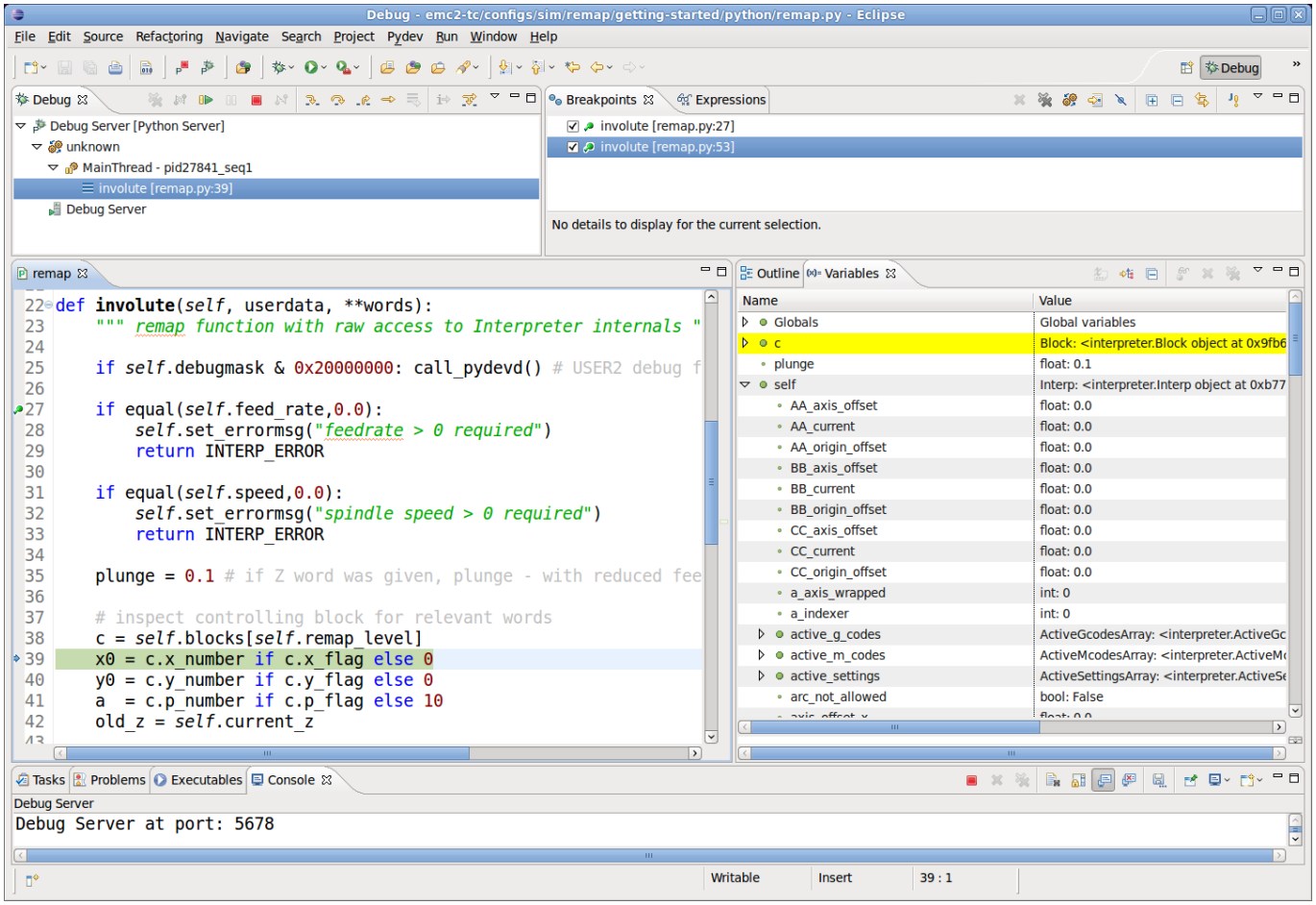
- call `pydevd.settrace()` in your module at some point to connect to the Eclipse Python debug server - here you can set breakpoints in your code, inspect variables, step etc as usual.

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.

11.13 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 during actual execution and 0 during preview. See *configs/sim/axis/remap/manual-toolchange-with-tool-length-switch/nc_subroutines/manual_change.ngc* for a complete usage example.

Within Embedded Python, the task instance can be checked for by testing *self.task* - this will be 1 in the milltask instance, and 0 in the preview instance(s).

11.14 Remappable Codes

11.14.1 Existing codes which can be remapped

The current set of **existing** codes open to redefinition is:

- Tx (Prepare)
- M6 (Change tool)
- M61 (Set tool number)
- M0 (pause a running program temporarily)
- M1 (pause a running program temporarily if the optional stop switch is on)
- M60 (exchange pallet shuttles and then pause a running program temporarily)
- S (set spindle speed)
- F (set feed)

Note that the use of M61 currently requires the use of iocontrol-v2.

11.14.2 Currently unallocated G-codes:

These codes are currently undefined in the current implementation of LinuxCNC and may be used to define new G-codes:

FIXTHIS too verbose

G0.1 G0.2 G0.3 G0.4 G0.5 G0.6 G0.7 G0.8 G0.9 G1.1 G1.2 G1.3 G1.4 G1.5 G1.6 G1.7 G1.8 G1.9 G2.1 G2.2 G2.3 G2.4 G2.5 G2.6 G2.7 G2.8 G2.9 G3.1 G3.2 G3.3 G3.4 G3.5 G3.6 G3.7 G3.8 G3.9 G4.1 G4.2 G4.3 G4.4 G4.5 G4.6 G4.7 G4.8 G4.9 G5.4 G5.5 G5.6 G5.7 G5.8 G5.9 G6 G6.1 G6.2 G6.3 G6.4 G6.5 G6.6 G6.7 G6.8 G6.9 G7.1 G7.2 G7.3 G7.4 G7.5 G7.6 G7.7 G7.8 G7.9 G8.1 G8.2 G8.3 G8.4 G8.5 G8.6 G8.7 G8.8 G8.9 G9 G9.1 G9.2 G9.3 G9.4 G9.5 G9.6 G9.7 G9.8 G9.9 G10.1 G10.2 G10.3 G10.4 G10.5 G10.6 G10.7 G10.8 G10.9 G11 G11.1 G11.2 G11.3 G11.4 G11.5 G11.6 G11.7 G11.8 G11.9 G12 G12.1 G12.2 G12.3 G12.4 G12.5 G12.6 G12.7 G12.8 G12.9 G13 G13.1 G13.2 G13.3 G13.4 G13.5 G13.6 G13.7 G13.8 G13.9 G14 G14.1 G14.2 G14.3 G14.4 G14.5 G14.6 G14.7 G14.8 G14.9 G15 G15.1 G15.2 G15.3 G15.4 G15.5 G15.6 G15.7 G15.8 G15.9 G16 G16.1 G16.2 G16.3 G16.4 G16.5 G16.6 G16.7 G16.8 G16.9 G17.2 G17.3 G17.4 G17.5 G17.6 G17.7 G17.8 G17.9 G18.2 G18.3 G18.4 G18.5 G18.6 G18.7 G18.8 G18.9 G19.2 G19.3 G19.4 G19.5 G19.6 G19.7 G19.8 G19.9 G20.1 G20.2 G20.3 G20.4 G20.5 G20.6 G20.7 G20.8 G20.9 G21.1 G21.2 G21.3 G21.4 G21.5 G21.6 G21.7 G21.8 G21.9 G22 G22.1 G22.2 G22.3 G22.4 G22.5 G22.6 G22.7 G22.8 G22.9 G23 G23.1 G23.2 G23.3 G23.4 G23.5 G23.6 G23.7 G23.8 G23.9 G24 G24.1 G24.2 G24.3 G24.4

G24.5 G24.6 G24.7 G24.8 G24.9 G25 G25.1 G25.2 G25.3 G25.4 G25.5 G25.6 G25.7 G25.8 G25.9 G26 G26.1 G26.2 G26.3 G26.4 G26.5 G26.6 G26.7 G26.8 G26.9 G27 G27.1 G27.2 G27.3 G27.4 G27.5 G27.6 G27.7 G27.8 G27.9 G28.2 G28.3 G28.4 G28.5 G28.6 G28.7 G28.8 G28.9 G29 G29.1 G29.2 G29.3 G29.4 G29.5 G29.6 G29.7 G29.8 G29.9 G30.2 G30.3 G30.4 G30.5 G30.6 G30.7 G30.8 G30.9 G31 G31.1 G31.2 G31.3 G31.4 G31.5 G31.6 G31.7 G31.8 G31.9 G32 G32.1 G32.2 G32.3 G32.4 G32.5 G32.6 G32.7 G32.8 G32.9 G33.2 G33.3 G33.4 G33.5 G33.6 G33.7 G33.8 G33.9 G34 G34.1 G34.2 G34.3 G34.4 G34.5 G34.6 G34.7 G34.8 G34.9 G35 G35.1 G35.2 G35.3 G35.4 G35.5 G35.6 G35.7 G35.8 G35.9 G36 G36.1 G36.2 G36.3 G36.4 G36.5 G36.6 G36.7 G36.8 G36.9 G37 G37.1 G37.2 G37.3 G37.4 G37.5 G37.6 G37.7 G37.8 G37.9 G38 G38.1 G38.6 G38.7 G38.8 G38.9 G39 G39.1 G39.2 G39.3 G39.4 G39.5 G39.6 G39.7 G39.8 G39.9 G40.1 G40.2 G40.3 G40.4 G40.5 G40.6 G40.7 G40.8 G40.9 G41.2 G41.3 G41.4 G41.5 G41.6 G41.7 G41.8 G41.9 G42.2 G42.3 G42.4 G42.5 G42.6 G42.7 G42.8 G42.9 G43.2 G43.3 G43.4 G43.5 G43.6 G43.7 G43.8 G43.9 G44 G44.1 G44.2 G44.3 G44.4 G44.5 G44.6 G44.7 G44.8 G44.9 G45 G45.1 G45.2 G45.3 G45.4 G45.5 G45.6 G45.7 G45.8 G45.9 G46 G46.1 G46.2 G46.3 G46.4 G46.5 G46.6 G46.7 G46.8 G46.9 G47 G47.1 G47.2 G47.3 G47.4 G47.5 G47.6 G47.7 G47.8 G47.9 G48 G48.1 G48.2 G48.3 G48.4 G48.5 G48.6 G48.7 G48.8 G48.9 G49.1 G49.2 G49.3 G49.4 G49.5 G49.6 G49.7 G49.8 G49.9 G50 G50.1 G50.2 G50.3 G50.4 G50.5 G50.6 G50.7 G50.8 G50.9 G51 G51.1 G51.2 G51.3 G51.4 G51.5 G51.6 G51.7 G51.8 G51.9 G52 G52.1 G52.2 G52.3 G52.4 G52.5 G52.6 G52.7 G52.8 G52.9 G53.1 G53.2 G53.3 G53.4 G53.5 G53.6 G53.7 G53.8 G53.9 G54.1 G54.2 G54.3 G54.4 G54.5 G54.6 G54.7 G54.8 G54.9 G55.1 G55.2 G55.3 G55.4 G55.5 G55.6 G55.7 G55.8 G55.9 G56.1 G56.2 G56.3 G56.4 G56.5 G56.6 G56.7 G56.8 G56.9 G57.1 G57.2 G57.3 G57.4 G57.5 G57.6 G57.7 G57.8 G57.9 G58.1 G58.2 G58.3 G58.4 G58.5 G58.6 G58.7 G58.8 G58.9 G59.4 G59.5 G59.6 G59.7 G59.8 G59.9 G60 G60.1 G60.2 G60.3 G60.4 G60.5 G60.6 G60.7 G60.8 G60.9 G61.2 G61.3 G61.4 G61.5 G61.6 G61.7 G61.8 G61.9 G62 G62.1 G62.2 G62.3 G62.4 G62.5 G62.6 G62.7 G62.8 G62.9 G63 G63.1 G63.2 G63.3 G63.4 G63.5 G63.6 G63.7 G63.8 G63.9 G64.1 G64.2 G64.3 G64.4 G64.5 G64.6 G64.7 G64.8 G64.9 G65 G65.1 G65.2 G65.3 G65.4 G65.5 G65.6 G65.7 G65.8 G65.9 G66 G66.1 G66.2 G66.3 G66.4 G66.5 G66.6 G66.7 G66.8 G66.9 G67 G67.1 G67.2 G67.3 G67.4 G67.5 G67.6 G67.7 G67.8 G67.9 G68 G68.1 G68.2 G68.3 G68.4 G68.5 G68.6 G68.7 G68.8 G68.9 G69 G69.1 G69.2 G69.3 G69.4 G69.5 G69.6 G69.7 G69.8 G69.9 G70 G70.1 G70.2 G70.3 G70.4 G70.5 G70.6 G70.7 G70.8 G70.9 G71 G71.1 G71.2 G71.3 G71.4 G71.5 G71.6 G71.7 G71.8 G71.9 G72 G72.1 G72.2 G72.3 G72.4 G72.5 G72.6 G72.7 G72.8 G72.9 G73.1 G73.2 G73.3 G73.4 G73.5 G73.6 G73.7 G73.8 G73.9 G74 G74.1 G74.2 G74.3 G74.4 G74.5 G74.6 G74.7 G74.8 G74.9 G75 G75.1 G75.2 G75.3 G75.4 G75.5 G75.6 G75.7 G75.8 G75.9 G76.1 G76.2 G76.3 G76.4 G76.5 G76.6 G76.7 G76.8 G76.9 G77 G77.1 G77.2 G77.3 G77.4 G77.5 G77.6 G77.7 G77.8 G77.9 G78 G78.1 G78.2 G78.3 G78.4 G78.5 G78.6 G78.7 G78.8 G78.9 G79 G79.1 G79.2 G79.3 G79.4 G79.5 G79.6 G79.7 G79.8 G79.9 G80.1 G80.2 G80.3 G80.4 G80.5 G80.6 G80.7 G80.8 G80.9 G81.1 G81.2 G81.3 G81.4 G81.5 G81.6 G81.7 G81.8 G81.9 G82.1 G82.2 G82.3 G82.4 G82.5 G82.6 G82.7 G82.8 G82.9 G83.1 G83.2 G83.3 G83.4 G83.5 G83.6 G83.7 G83.8 G83.9 G84.1 G84.2 G84.3 G84.4 G84.5 G84.6 G84.7 G84.8 G84.9 G85.1 G85.2 G85.3 G85.4 G85.5 G85.6 G85.7 G85.8 G85.9 G86.1 G86.2 G86.3 G86.4 G86.5 G86.6 G86.7 G86.8 G86.9 G87.1 G87.2 G87.3 G87.4 G87.5 G87.6 G87.7 G87.8 G87.9 G88.1 G88.2 G88.3 G88.4 G88.5 G88.6 G88.7 G88.8 G88.9 G89.1 G89.2 G89.3 G89.4 G89.5 G89.6 G89.7 G89.8 G89.9 G90.2 G90.3 G90.4 G90.5 G90.6 G90.7 G90.8 G90.9 G91.2 G91.3 G91.4 G91.5 G91.6 G91.7 G91.8 G91.9 G92.4 G92.5 G92.6 G92.7 G92.8 G92.9 G93.1 G93.2 G93.3 G93.4 G93.5 G93.6 G93.7 G93.8 G93.9 G94.1 G94.2 G94.3 G94.4 G94.5 G94.6 G94.7 G94.8 G94.9 G95.1 G95.2 G95.3 G95.4 G95.5 G95.6 G95.7 G95.8 G95.9 G96.1 G96.2 G96.3 G96.4 G96.5 G96.6 G96.7 G96.8 G96.9 G97.1 G97.2 G97.3 G97.4 G97.5 G97.6 G97.7 G97.8 G97.9 G98.1 G98.2 G98.3 G98.4 G98.5 G98.6 G98.7 G98.8 G98.9 G99.1 G99.2 G99.3 G99.4 G99.5 G99.6 G99.7 G99.8 G99.9

11.14.3 Currently unallocated M-codes:

These codes are currently undefined in the current implementation of LinuxCNC and may be used to define new M-codes:

```
M10
M11 M12 M13 M14 M15 M16 M17 M18 M19 M20
M21 M22 M23 M24 M25 M26 M27 M28 M29 M31 M32 M33 M34 M35 M36 M37 M38 M39 M40
M41 M42 M43 M44 M45 M46 M47 M54 M55 M56 M57 M58 M59 M74 M75 M76 M77 M78 M79 M80
M81 M82 M83 M84 M85 M86 M87 M88 M89 M90
M91 M92 M93 M94 M95 M96 M97 M98 M99
```

All codes between `M199` and `M999`.

11.14.4 readahead time and execution time

foo

11.14.5 plugin/pickle hack

foo

11.14.6 Module, methods, classes, etc reference

foo

11.15 Introduction: Extending Task Execution

foo

11.15.1 Why would you want to change Task Execution?

foo

11.15.2 A diagram: task, interp, iocontrol, UI (??)

foo

11.16 Models of Task execution

foo

11.16.1 Traditional iocontrol/iocontrolv2 execution

foo

11.16.2 Redefining IO procedures

foo

11.16.3 Execution-time Python procedures

foo

11.17 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.

11.17.1 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
4. interpreter execution state

(3) 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) 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).

11.17.2 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*.

11.17.3 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 Gcode, 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.

11.17.4 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 O100 if [#5070 EQ 0]
N50    G1 Y20 Z-5
N60 O100 else
N70    G0 Z30
N80 O100 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.x: final position and success/failure not predictable

11.17.5 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_FINISH*).
- 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.

11.17.6 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).

11.17.7 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.

11.17.8 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.

11.17.9 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).

11.17.10 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.

11.17.10.1 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.

11.17.11 How Tx (Prepare Tool) works

11.17.11.1 Interpreter action on a Tx command

All the interpreter does is evaluate the toolnumber paramter, looks up its corresponding pocket, remembers it in the `selected_pocket` variable for later, and queues a canon command (SELECT_POCKET). See *Interp::convert_tool_select* in *src/emc/rs274/interp_execute.cc*.

11.17.11.2 Task action on SELECT_POCKET

When task gets around to handle a SELECT_POCKET, it sends a EMC_TOOL_PREPARE 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 - it defeats the idea that changer preparation and code execution can proceed in parallel.

11.17.11.3 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* istn strictly necessary IMO)

11.17.11.4 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`.

11.17.12 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*.

11.17.12.1 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.

11.17.12.2 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.

11.17.12.3 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.

11.17.12.4 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`.

11.17.13 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*.

11.17.13.1 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`.

11.18 Optional Interpreter features: ini file configuration

There are some interpreter features in this branch which are experimental, and not backwards compatible, which is why they need to be enabled explicitly. They are specified as follows:

```
[RS274NGC]
FEATURES = <feature mask>
```

Mask bits are:

Retain G43:1 (experimental)
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. This is experimental as it changes the operation of legal ngc program, but it could be argued that those programs are buggy or likely to be not what the author intended.

add n_args parameter:2
A called subroutine can determine the number of actual positional parameters passed by inspecting the `#<n_args>` parameter.

enable #<_ini[section]name> read only variables:4
if set, the interpreter will fetch read-only values from the ini file through this special variable syntax.

enable #<_hal[Hal item]> read only variables:8
if set, the interpreter will fetch read-only values from HAL file through this special variable syntax.

preserve case in O-word names within comments:16
if set, enables reading of mixed-case HAL items in structured comments like *(debug, #<_hal[MixedCaseItem])*. Really a kludge which should go away.

11.19 Named parameters and inifile variables

To access ini file values from G-code, use the following named parameter syntax:

`#<_ini[section]name>`

For example, if the ini file looks like so:

```
[SETUP]
XPOS = 3.145
YPOS = 2.718
```

you may refer to the O-word 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 inifile 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.

Permanent setup information is usually stored in the ini file. While ini variables can be easily accessed from the shell, Python and C code, so far there was no way to refer to ini file variables from G-code. This release enables such access. The feature was motivated by the need to replace ini variables which are currently used in the hard-coded tool change process, like the `[EMCIO]TOOL_CHANGE_POSITION` parameter.

Caution
this section doesn't really belong here but since it comes with the same branch, here it rests for now until its clear this will be merged. It should go into the gcode/overview Named Parameters section.

11.20 Named parameters and HAL items

The variables are read during read-ahead and should not be used for run time evaluation of *current position* or other execution time variables.

To read arbitrary HAL pins, signals and parameters from G-code, use the following named parameter syntax:

`#<_hal[hal_name]>`

where `hal_name` may be a pin, parameter or signal name.

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-mmemonic namespace and is unnecessary cumbersome just as a UI/Interpreter communications mechanism.

Note
The values are are only updated when the G code is not running.

Caution
this section doesn't really belong here but since it comes with the same branch, here it rests for now until its clear this will be merged. It should go into the gcode/overview Named Parameters section.

11.21 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.

11.22 Build notes - Lucid (10.04)

For the interpreter & task Python plug ins, this is required:

```
apt-get install libboost-python1.40-dev
```

When compiling you might notice that interpmodule.cc takes very long to compile, which is normal - the extensive use of C++ templates makes the compiler breathe heavily.

If you want to play with the configs/sim/axis/remap/iocontrol-removed example, you need to install as follows:

```
apt-get unixODBC-dev libsqliteodbc sqlite3
git clone https://code.google.com/p/pyodbc/
sudo  python setup.py build install
```

If you'd want to try how the Firefox SQlite manager plugin looks & feels as a tool table editor, try this:

1. read http://code.google.com/p/sqlite-manager/
2. download the zip file *SQLiteManager 0.7.7 as XULRunner App* or whatever is the latest from http://code.google.com/p/-sqlite-manager/downloads/list
3. create a directory under your home directory, eg ~/sqlite-manager
4. unzip the zp file from 1) into this directory

try running firefox with this plug in and the tooltable.sqlite file in this directory like so:

```
`firefox -app <homdir>/sqlite-manager/application.ini -f tooltable.sqlite`

firefox should come up with the sqlite manager extension and having this database ↩
    opened
```

1. adapt the following command line with appropriate paths in the ini file:

```
TOOL_EDITOR=firefox -app /home/mah/sqlite-manager/application.ini -f /home/mah/emc2-dev/
configs/sim/remap/iocontrol-removed/tooltable.sqlite
```

11.23 Build notes - Hardy (8.04)

Building and running on Hardy is possible. run tests works fine too, so the remapping framework per se is ok.

However running the examples is quite limited as of now:

The Git version included in 8.04 is too old to pull https://code.google.com/p/pyodbc/ so the iocontrol-removed demo cant be run.

The python-gtkglext1 dependency is missing for reasons I dont understand.

Even if python-gtkglext1 is installed, the startup of the manualtoolchange and racktoolchange demos fails due to gladevcp startup issues.

Note that has nothing to to with the new code but rather the very old platform trying to run gladevcp.

11.24 Workarounds

The workaround mentioned below was necessary up to commit d21a488a9e82dd85aa17207b80e3d930afeff202 . References to DISPLAY_LD_PRELOAD and TASK_LD_PRELOAD have been removed from the ini files under configs/sim/axis/remap because they are not needed anymore.

Configure now tests whether a workaround is required, and automatically does *the right thing* if needed.

```
# Michael Haberler 4/2011
#
# if you get a segfault like described
# here: https://bugs.launchpad.net/ubuntu/+source/mesa/+bug/259219
# or here: https://www.libavg.de/wiki/LinuxInstallIssues#glibc_invalid_pointer :
#
# specify a workaround with:
# [DISPLAY]
# DISPLAY_LD_PRELOAD = /usr/lib/libstdc++.so.6
# and
# [TASK]
# TASK_LD_PRELOAD = /usr/lib/libstdc++.so.6
#
# this is actually a bug in libgl1-mesa-dri and it looks
# it has been fixed in mesa - 7.10.1-0ubuntu2
# unfortunately for now this workaround is needed
DISPLAY_LD_PRELOAD = /usr/lib/libstdc++.so.6
```

11.25 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.

Chapter 12

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.

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)

12.1 Modifying an existing configuration

An existing configuration can be modified to use moveoff_gui as follows:

1. Make inifile entries for HALUI and LIB:hookup_moveoff.tcl. The entry for hookup_moveoff.tcl should follow HALFILES that connect the pins for axis.N.motor-pos-cmd, axis.N.motor-pos-fb, and components connected to these pins (pids and encoders typically).

```
[HAL]
HALUI = halui
...
HALFILE = LIB:hookup_moveoff.tcl
```

1. Add inifile 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 is used(NOT RECOMMENDED)):

```
[MOVEOFF_n]
MAX_LIMIT =
MIN_LIMIT =
MAX_VELOCITY =
MAX_ACCELERATION =
```

1. Add inifile entries for moveoff component settings (omit to use moveoff defaults):

```
[MOVEOFF]
EPSILON =
WAYPOINT_SAMPLE_SECS =
WAYPOINT_THRESHOLD =
```

1. Add inifile entries to start the gui:

```
[APPLICATIONS]
# Note: a delay may be required if there are [HAL]POSTGUI_HALFILE dependencies
DELAY = 0
APP = moveoff_gui option1 option2 ...
```

For details on available Options, Use:

```
$ 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: not used)
                             (resume program when move-enable deasserted)
   [-delay delay_secs]     (default: 5 (resume delay))

Options for special cases:
   [-noentry]              (default: not used)
                             (don't create entry widgets)
   [-no_resume_inhibit]    (default: not used)
                             (do not use a resume-inhibit-pin)
   [-no_pause_requirement] (default: not used)
                             (no check for halui.program.is-paused)
```

The moveoff_gui will provide a display and and control for enabling offsetting if the pin mv.move-enable is NOT connected when moveoff_gui is started.

If the mv.move-enable pin is connected when moveoff_gui is started, the gui will provide a display but no controls. This mode supports Hal connections for a jog wheel or other methods of controlling the enable and offset input pins (mv.move-enable, mv.offset-M, mv.backtrack-enable).

If the halfile LIB:hookup_moveoff.tcl is used to load and connect the moveoff component, the mv.move-enable pin is not connected and local controls provided by moveoff_gui will be used. To enable external controls, subsequent halfiles should connect the mv.move-enable pin. For example, the configs/sim/* demo configurations use a simple halfile to connect the mv.move-enable,mv.offset-in-M, and mv.bactrack-enable pins to signals:

```
[HAL]
HALUI = halui
...
HALFILE = LIB:hookup_moveoff.tcl
HALFILE = LIB:moveoff_external.hal
```

Part III

GUI

Chapter 13

Python Virtual Control Panel

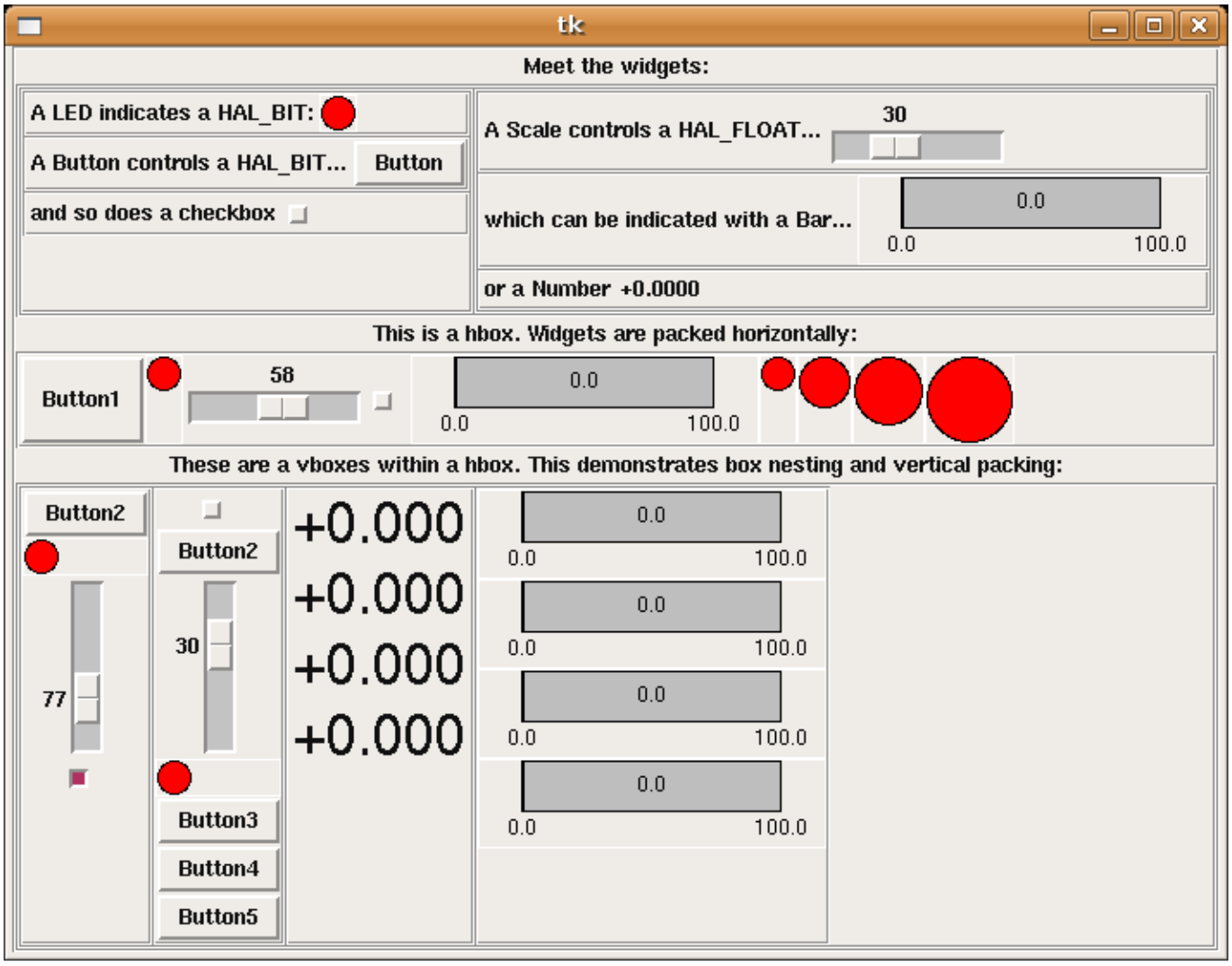
13.1 Introduction

Python Virtual Control Panel The PyVCP (Python Virtual Control Panel) 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.



13.2 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>
```

If you place this text in a file called tiny.xml, and run

```
halrun -I 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.

13.3 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.

13.4 AXIS

Since AXIS uses the same GUI toolkit (Tkinter) as PyVCP, it is possible to include a PyVCP panel on the right side of the normal 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 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
```

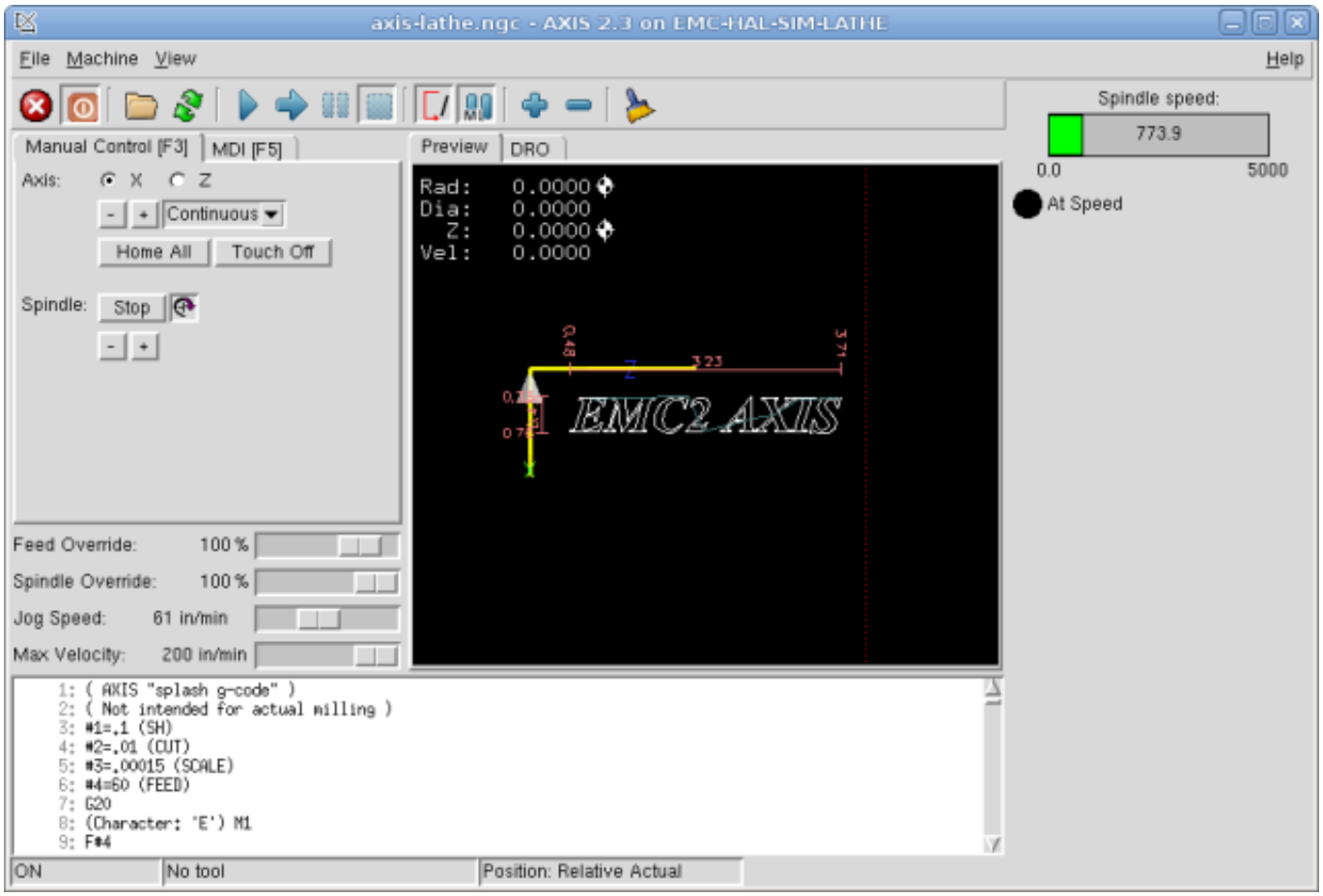
To make our widget actually display the spindle-speed it needs to be hooked up to the appropriate HAL signal. A .hal file that will be run once AXIS and PyVCP have started can be specified in the [HAL] section of the .ini file:

```
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.

13.5 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 panelname 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:

```
waituser 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.

13.6 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 manual. 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.

1. Widgets for indicating *bit* signals: led, rectled
2. Widgets for controlling *bit* signals: button, checkbutton, radiobutton
3. Widgets for indicating *number* signals: number, s32, u32, bar, meter
4. Widgets for controlling *number* signals: spinbox, scale, jogwheel
5. Helper widgets: hbox, vbox, table, label, labelframe

13.6.1 Syntax

Each widget is described briefly, followed by the markup used, and a screen shot. All tags inside the main widget tag are optional.

13.6.2 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.

13.6.2.1 Comments

To add a comment use the xml syntax for a comment.

```
<!-- My Comment -->
```

13.6.2.2 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.

13.6.2.3 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

13.6.2.4 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 of the HAL manual.

13.6.3 Label

A label is a piece of text on your panel.

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>
    <disable>False</disable>
</label>
```

The above code produced this example.

13.6.4 Multi_Label

An extention 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.

```
<multilabel>
    <legends>["Label1" "Label2" "Label3" "Label4" "Label5" "Label6"]</legends>
    <font>("Helvetica",20)</font>
    <disable>False</disable>
</multilabel>
```

13.6.5 LEDs

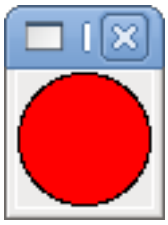
A LED is used to indicate the status of a *bit* halpin. The LED color will be on_color when the halpin is true, and off_color otherwise.

- <*halpin*> - sets the name of the pin, default is *led.n*, where n is an integer
- <*size*> - sets the size of the led, default is 20
- <*on_color*> - sets the color of the LED when the pin is true. default is *green*
- <*off_color*> - sets the color of the LED when the pin is false. default is *red*
- <*disable_pin*> - when true adds a disable pin to the led.
- <*disabled_color*> - sets the color of the LED when the pin is disabled.

13.6.5.1 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.



13.6.5.2 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.

13.6.6 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 formatting 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
- *<bg>"color"</bg>* - the color of the button

13.6.6.1 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
```

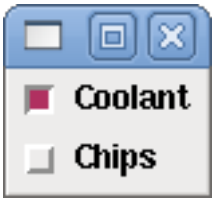
The above code produced this example.

13.6.6.2 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

```
<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. The coolant checkbutton is checked. Notice the extra spaces in the Chips text to keep the checkbuttons aligned.

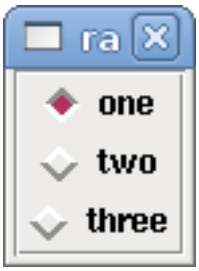


13.6.6.3 Radiobutton

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.



Note that the HAL pins in the example above will me named my-radio.one, my-radio.two, and my-radio.three. In the image above, *one* is the selected value.

13.6.7 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

13.6.7.1 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.

- <*font*> - 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>
```

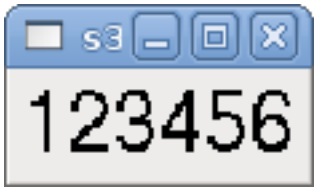
- <*format*> - is a *C-style* format specified that determines how the number is displayed.

13.6.7.2 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.



13.6.7.3 u32 Number

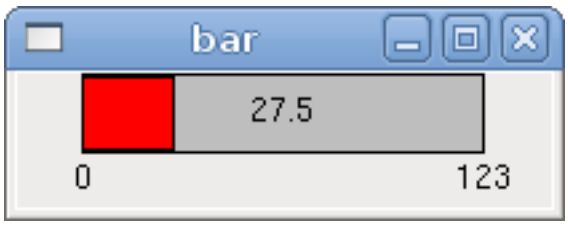
The u32 number widget displays the value of a u32 number. The syntax is the same as *number* except the name which is <u32>.

13.6.7.4 Bar

A bar widget displays the value of a FLOAT signal both graphically using a bar display and numerically. The colour of the bar can be set as one colour throughout its range (default using fillcolor) or set to change colour 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 colour)

```
<bar>
    <halpin>"my-bar"</halpin>
    <min_>0</min_>
    <max_>150</max_>
    <bgcolor>"grey"</bgcolor>
    <fillcolor>"red"</fillcolor>
    <range1>0,100,"green"</range1>
    <range2>101,135,"orange"</range1>
    <range3>136, 150,"red"</range1>
</bar>
```

The above code produced this example.

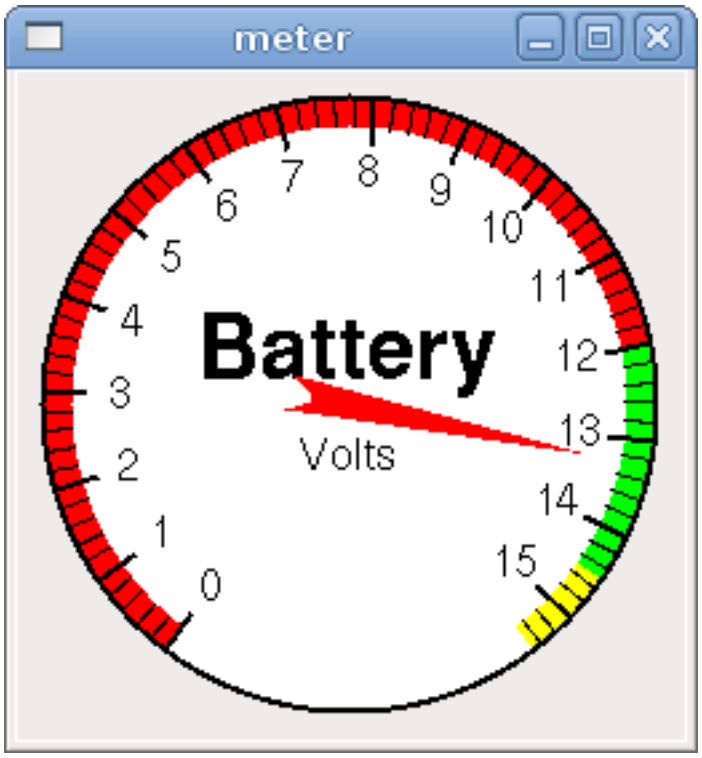


13.6.7.5 Meter

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.



13.6.8 Number Inputs

13.6.8.1 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>
```

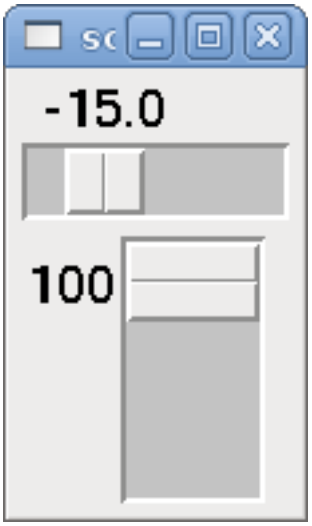
The above code produced this example.

13.6.8.2 Scale

Scale controls a float or a s32 pin. You increase or decrease the value of the pin be 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-vscale"</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.

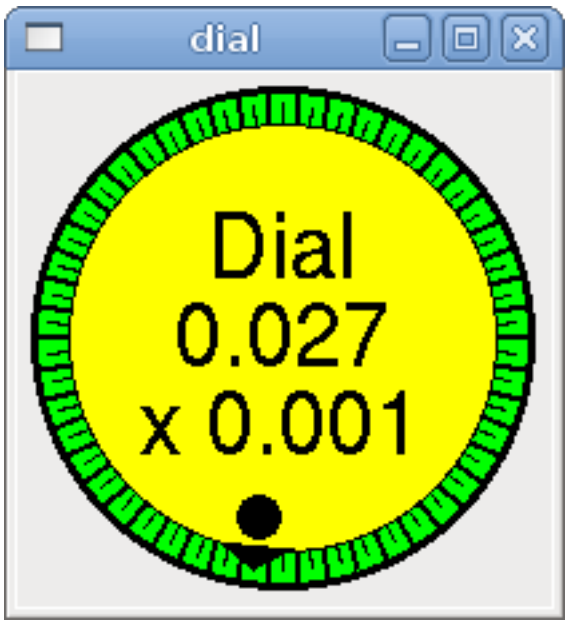


13.6.8.3 Dial

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.

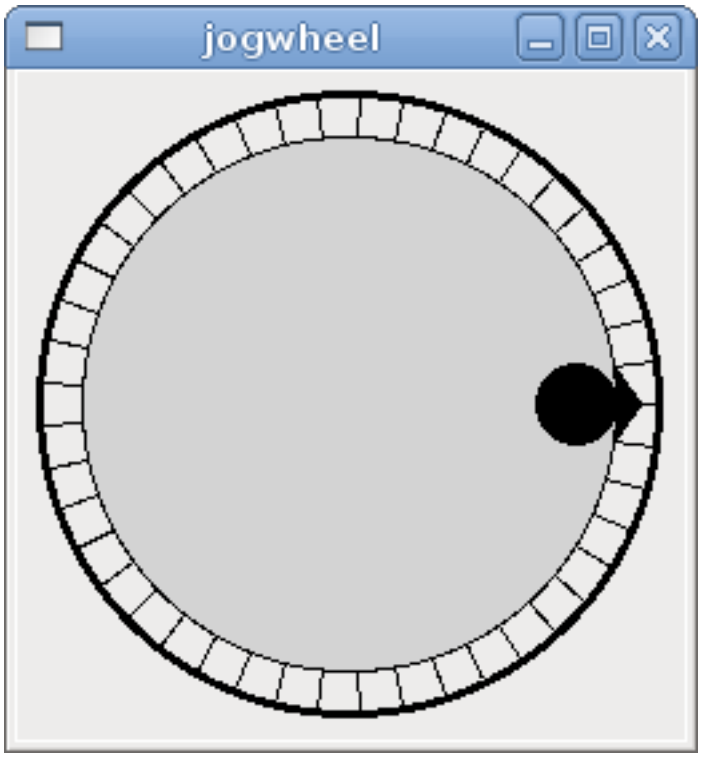


13.6.8.4 Jogwheel

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.

```
<jogwheel>
    <halpin>"my-wheel"</halpin>
    <cpr>45</cpr>
    <size>250</size>
</jogwheel>
```

The above code produced this example.



13.6.9 Images

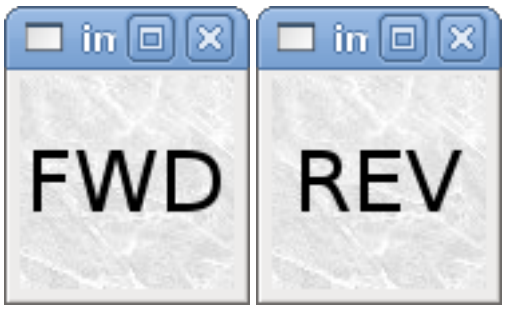
Image displays use only .gif image format. All of the images must be the same size. The images must be in the same directory as your ini file (or in the current directory if running from the command line with halrun/halcmd).

13.6.9.1 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.



13.6.9.2 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.

Notice that the default is the min even though it is set higher than max unless there is a negative min.

13.6.10 Containers

Containers are widgets that contain other widgets. Containers are used to group other widgets.

13.6.10.1 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>
```

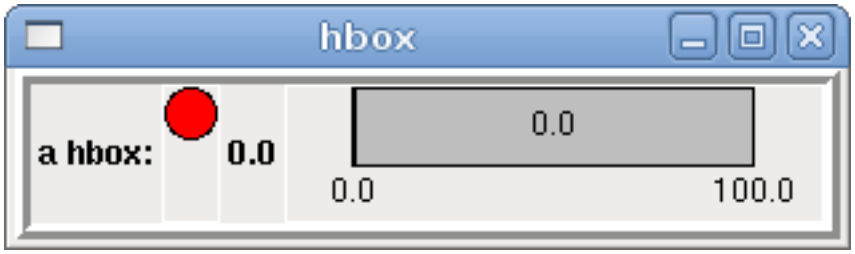
The above code produced this example.

13.6.10.2 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.



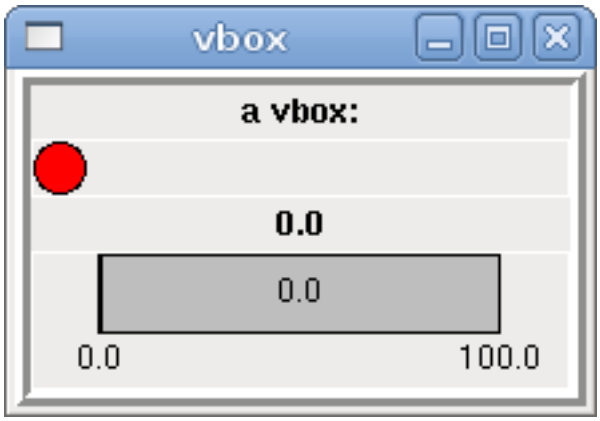
Inside an Hbox, you can use the <*boxfill fill=""*/>, <*boxanchor anchor=""*/>, and <*boxexpand expand=""*/> tags to choose how items in the box behave when the window is re-sized. For details of how fill, anchor, and expand behave, refer to the Tk *pack* manual page, *pack(3tk)*. By default, *fill="y"*, *anchor="center"*, *expand="yes"*.

13.6.10.3 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.



Inside a 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. For details of how fill, anchor, and expand behave, refer to the Tk *pack* manual page, *pack(3tk)*. By default, *fill="x"*, *anchor="center"*, *expand="yes"*.

13.6.10.4 Labelframe

A labelframe is a frame with a groove and a label at the upper-left corner.

```
<labelframe text="Group Title">
    <font>("Helvetica",16)</font>
    <hbox>
    <led/>
    <led/>
    </hbox>
</labelframe>
```

The above code produced this example.

13.6.10.5 Table

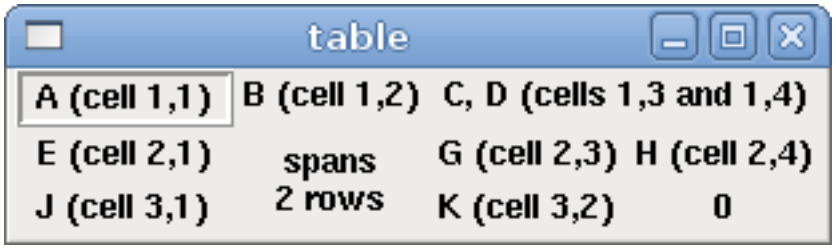
A table is a container that allows layout in a grid of rows and columns. Each row is started by a <*tablerow/*> tag. A contained widget may span rows or columns through the use of the <*tablespan rows= cols=/*> tag. The sides of the cells to which the contained widgets "stick" may be set through the use of the <*tablesticky sticky=/*> tag. A table expands on its flexible rows and columns.

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.



13.6.10.6 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.


tabs
Spindle
Green Eggs
Ham
Spindle speed:
3500.0
0.0
5000
tabs
Spindle
Green Eggs
Ham
(this is the green eggs tab)
tabs
Spindle
Green Eggs
Ham
(this tab has nothing on it)

Chapter 14

PyVCP Examples

14.1 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.

14.2 Floating

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 sppanel pyvcp -c sppanel 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.

14.3 Jog Buttons

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.

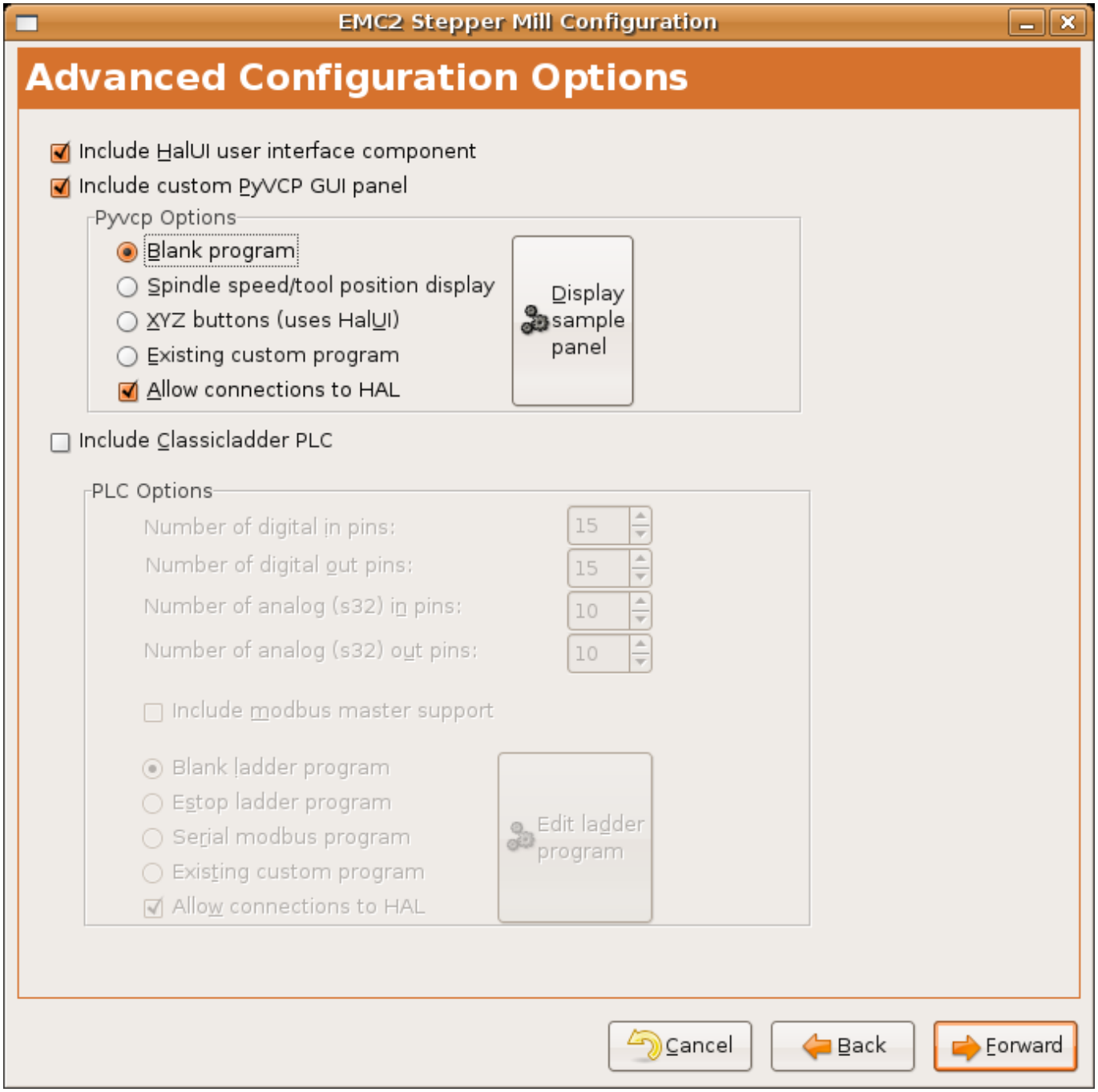


Figure 14.1: XYZ Wizard Configuration

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.

14.3.1 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.

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.

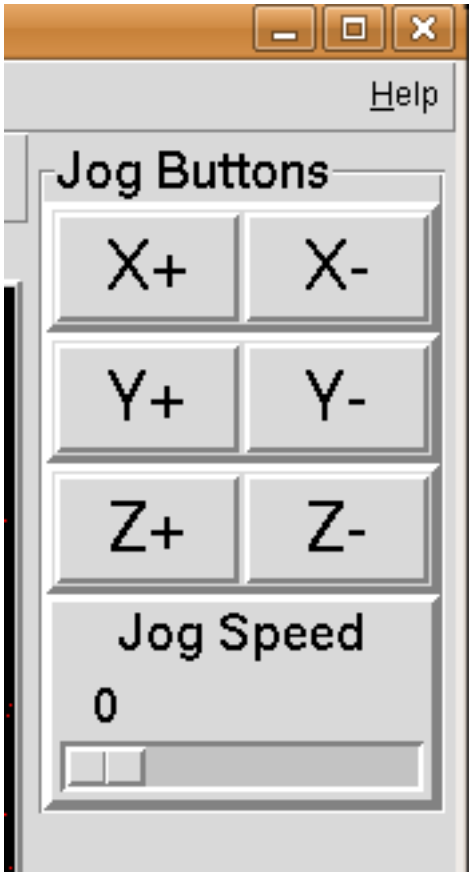


Figure 14.2: Jog Buttons

14.3.2 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.jog.0.minus <= pyvcp.x-minus
net my-jogxplus halui.jog.0.plus <= pyvcp.x-plus

# connect the Y PyVCP buttons
net my-jogyminus halui.jog.1.minus <= pyvcp.y-minus
net my-jogyplus halui.jog.1.plus <= pyvcp.y-plus

# connect the Z PyVCP buttons
net my-jogzminus halui.jog.2.minus <= pyvcp.z-minus
net my-jogzplus halui.jog.2.plus <= pyvcp.z-plus

# connect the PyVCP jog speed slider
  net my-jogspeed halui.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.

14.4 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.

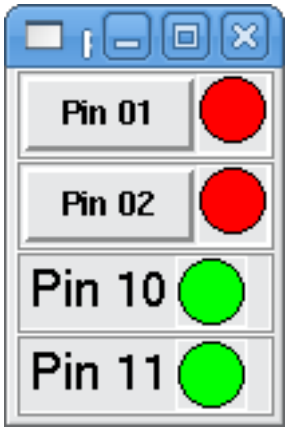


Figure 14.3: Port Tester Panel

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 name1=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.

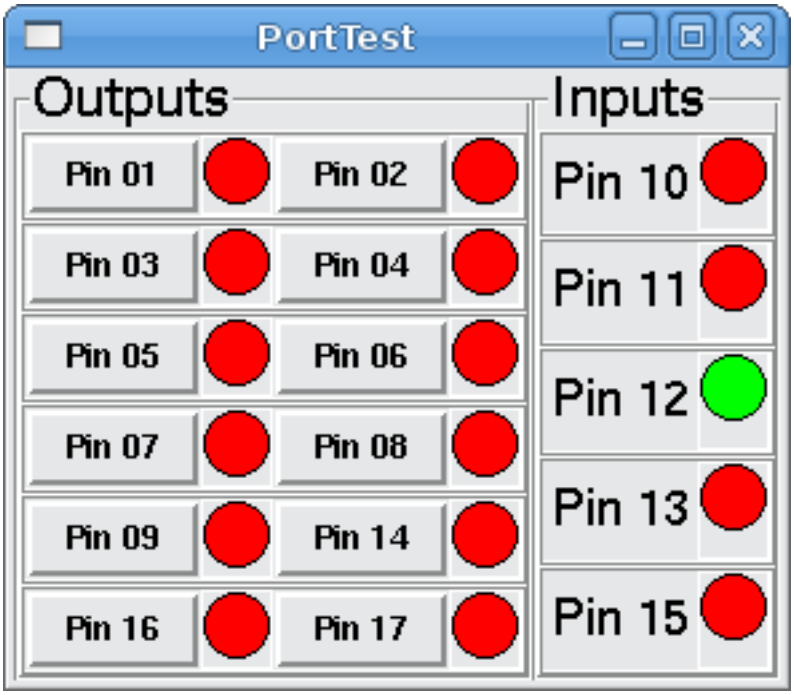


Figure 14.4: Port Tester Complete

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.

14.5 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.

14.5.1 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.

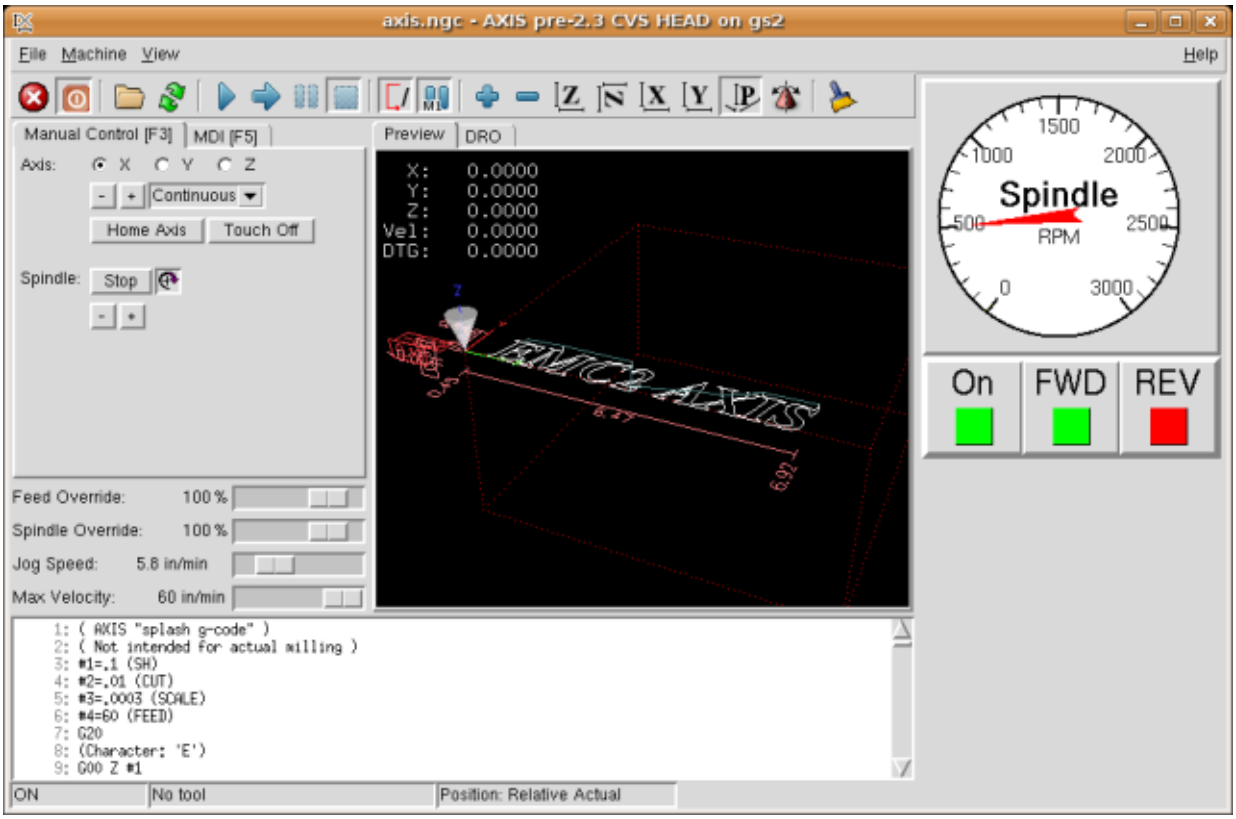


Figure 14.5: GS2 Panel

14.5.2 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.

Chapter 15

Glade Virtual Control Panel

15.1 What is GladeVCP?

GladeVCP is an 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 PyGTK 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.

15.1.1 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

15.2 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 iconifyed/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.

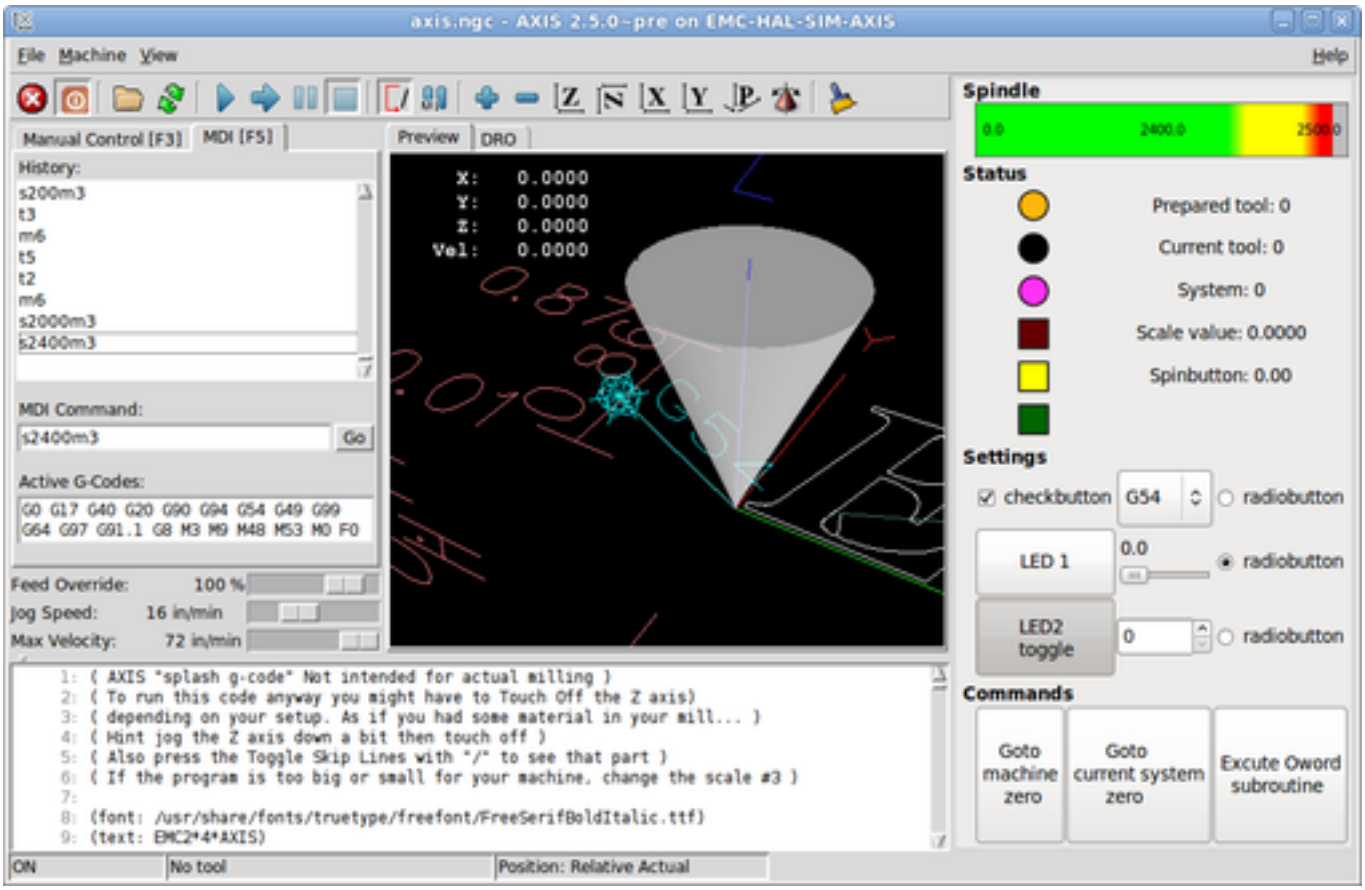
Git Checkout The following instructions only apply if you're using a git checkout. Open a terminal and change to the directory created by git then issue the commands as shown.

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.

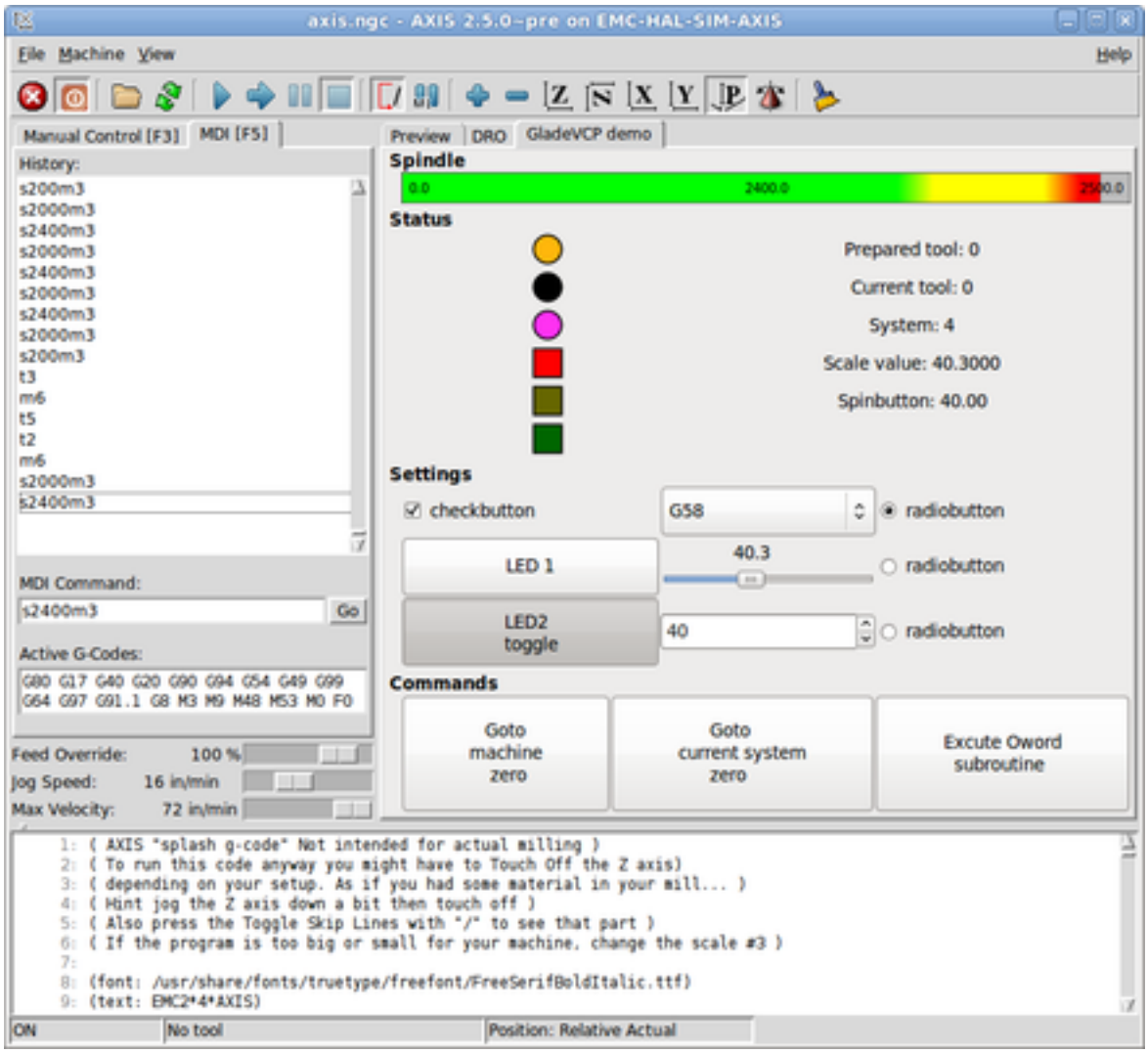
Run the sample GladeVCP panel integrated into Axis like PyVCP as follows:

```
$ cd configs/sim/axis/gladevcp
$ linuxcnc gladevcp_panel.ini
```



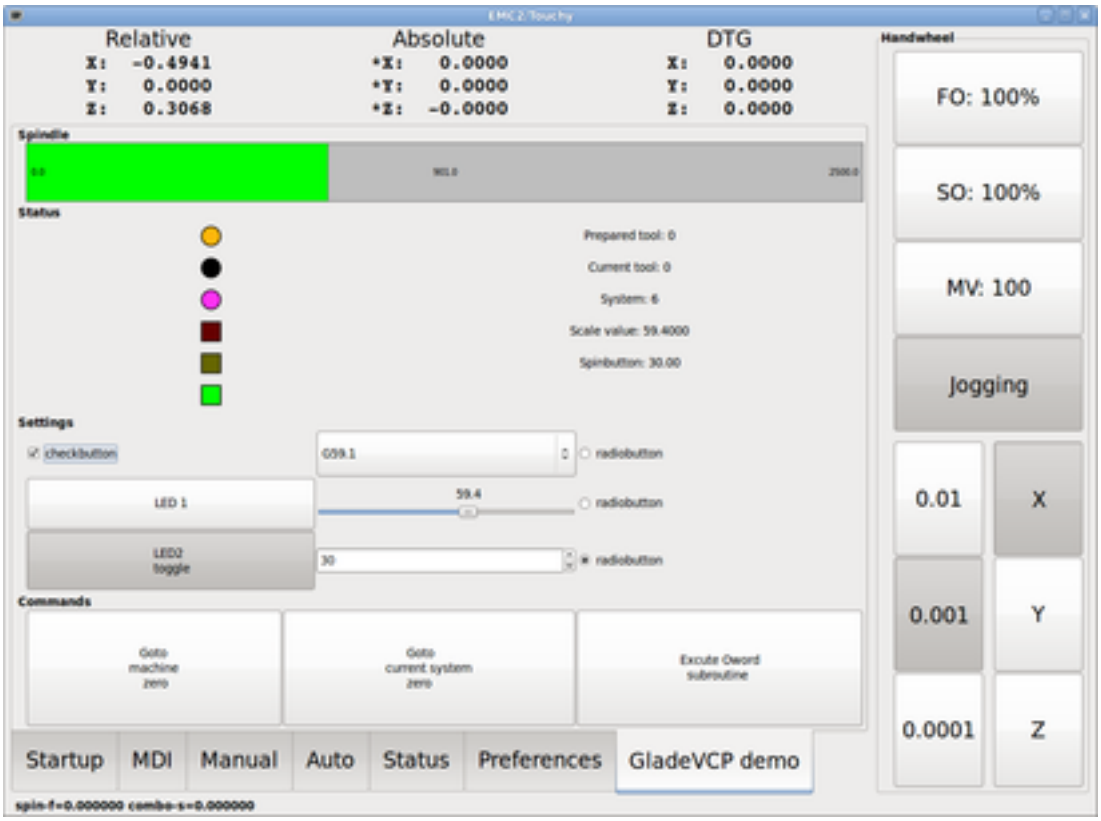
Run the same panel, but as a tab inside Axis:

```
$ cd configs/sim/axis/gladevcp
$ linuxcnc gladevcp_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.

15.2.1 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

15.2.2 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.

15.2.3 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

The is just touching upon the concept - the callback mechanism will be handled in more detail in the GladeVCP Programming section.

15.3 Creating and Integrating a Glade user interface

15.3.1 Prerequisite: Glade installation

To view or modify Glade UI files, you need glade installed - it is not needed just to run a GladeVCP panel. If the glade command is missing, install it with the command:

```
$ sudo apt-get install glade
```

Verify the version number to be greater than 3.6.7:

```
$ glade --version
glade3 3.6.7
```

(On recent systems, the required glade package is glade-gtk2)

15.3.2 Running Glade to create a new user interface

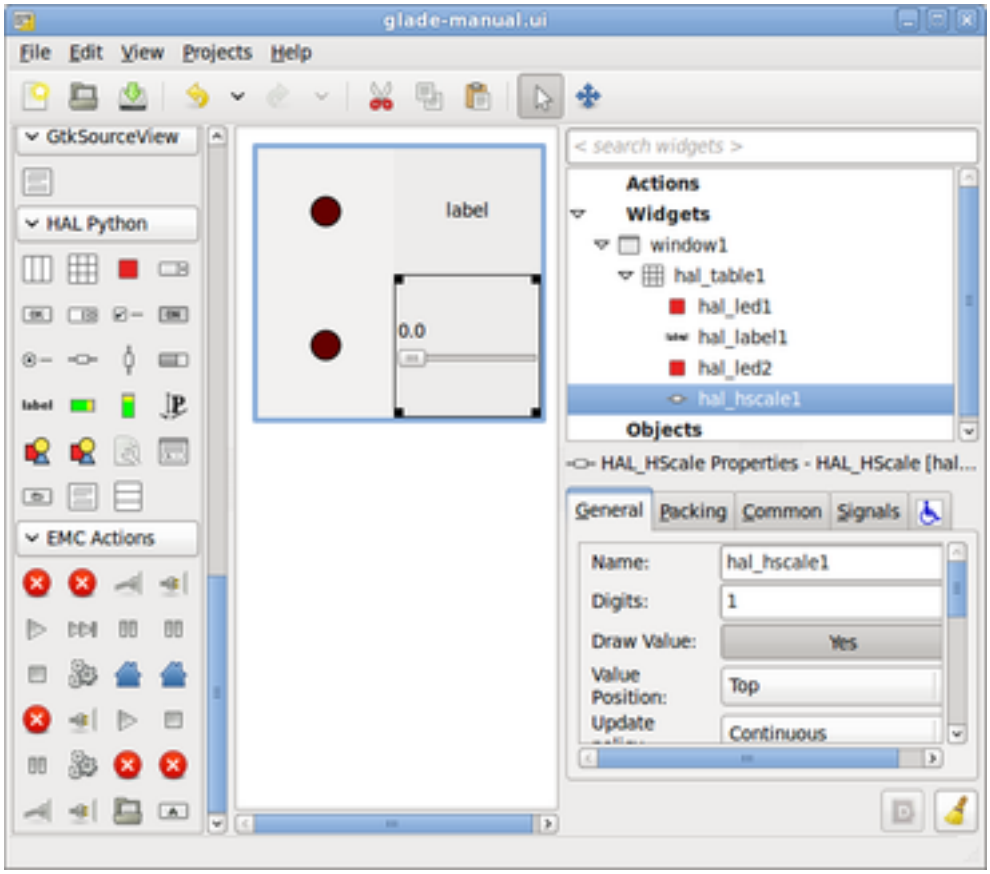
This section just outlines the initial LinuxCNC-specific steps. For more information and a tutorial on glade, see http://glade.gnome.org. Some glade tips & tricks may also be found on youtube.

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 *EMC 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*.

15.3.3 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 overriden 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`.

15.3.4 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.

15.3.5 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 Oword subs live here
SUBROUTINE_PATH = ../../nc_files/gladevcp_lib
```

The HAL component name of a GladeVCP application started with the GLADEVCP option is fixed: `gladevcp`. The command line actually run by Axis in the above configuration is as follows:

```
halcmd loadusr -Wn gladevcp gladevcp -c gladevcp -x {XID} <arguments to GLADEVCP>
```

This means you may add arbitrary gladevcp options here, as long as they dont collide with the above command line options.

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 you 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.

15.3.6 Integrating into Axis as a tab next to DRO and Preview

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 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.

15.3.7 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 Oword 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 you 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.

15.4 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 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

15.5 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 gladVCP 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 <halcmd file>* option (see previous section).

15.6 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
- 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 Gcode display and edit: HAL_Sourceview
- widget for MDI input and history display: MDI History

15.6.1 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>-s (s32) pin
- *HAL_ProgressBar*, which has a <widgetname>-value input pin, and a <widgetname>-scale input pin.

15.6.2 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.

The 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.

15.6.3 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!.

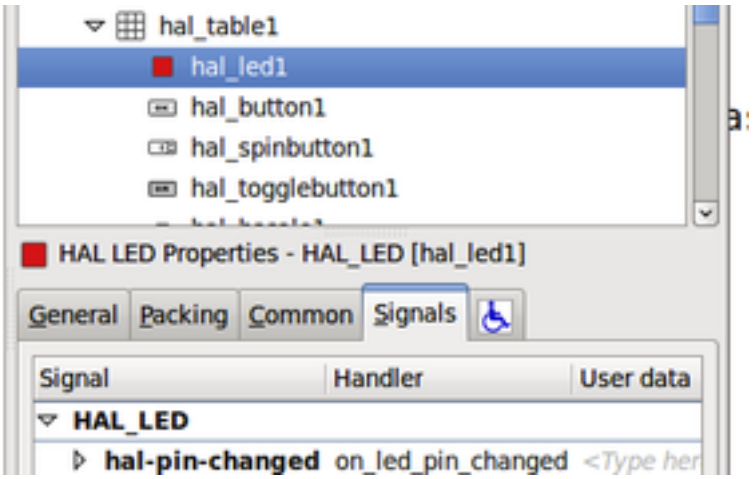
It might be tempting to *set HAL widget input pins* programmatically. Note this defeats the purpose of an input pin in the first place - it should be linked to, and react to signals generated by other HAL components. While there is currently no write protection on writing to input pins in HAL Python, this doesn't make sense. You might use setp pinname value in the associated halfile 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>,<type>,<direction>)` method (see GladeVCP Programming for an example).

15.6.4 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:

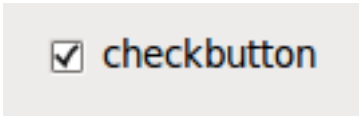


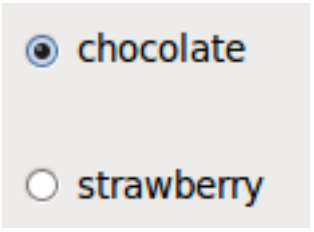
The example in `configs/apps/gladevcp/complex` shows how this is handled in Python.

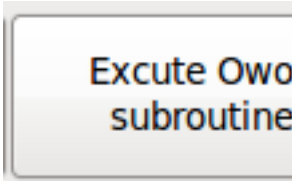
15.6.5 Buttons

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`

Check button: 

Radio buttons: 

Toggle button: 

.

Tip
Defining radio button groups in Glade:

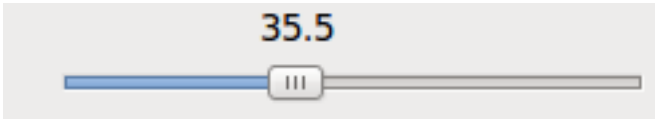
- decide on default active button
- in the other button's *General→Group* select the default active button's name in the *Choose a Radio Button in this project* dialog.

See `configs/apps/gladevcp/by-widget/` for a GladeVCP applications and UI file for working with radio buttons.

15.6.6 Scales

HAL_HScale and HAL_VScale are derived from the GtkHScale and GtkVScale respectively. They have one output FLOAT pin with name equal to widget name. Scales have no additional properties.

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.

Example HAL_HScale: 

.

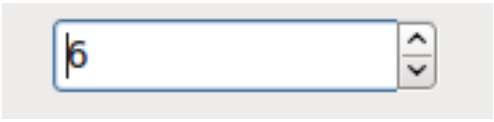
15.6.7 SpinButton

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.

Example SpinButton: 

.

15.6.8 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 degrease 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

It 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 witch 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

Direct program control

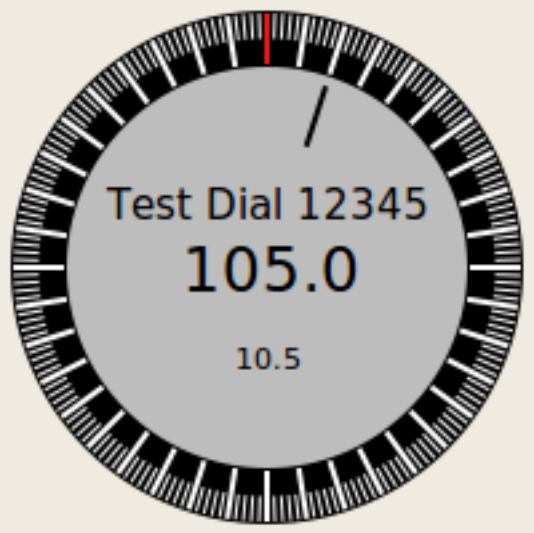
```
There are ways to directly control the widget using Python.

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.
```

Example Hal_Dial:



15.6.9 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 degrease the counts.

- Counterclockwise = reduce counts
- Clockwise = increase counts
- Wheel up = increase counts

- Wheel down = reduce counts

As moving the mouse the drag and drob way may be faster than the widget can update itself, you may loose counts turning to fast. It is recommended to use the mouse wheel, and only for very rough movements the drag and drob way.

JogWheel exports it's count value as hal pin:

<widgetname>-s
out S32 pin

It 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 witch 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.

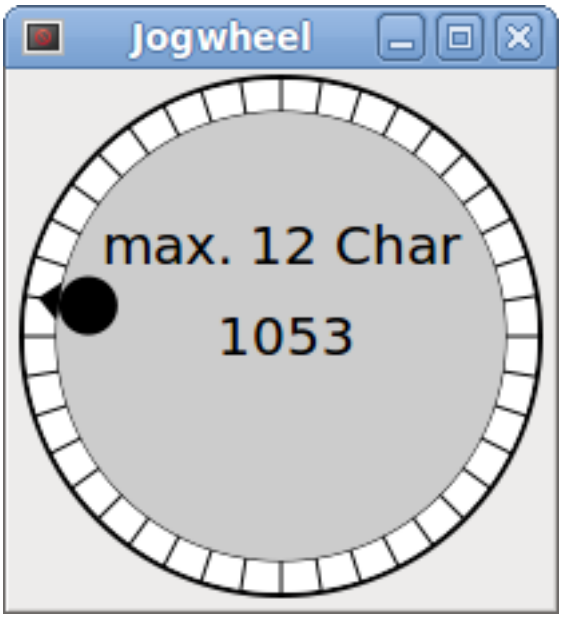
Direct program control

```
There a couple ways to directly control the widget using Python.

Using goobject 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"
```

Example JogWheel:



15.6.10 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 Pythons format() method.
Example: `Distance:%.03f` will display the text and the pin value with 3 fractional digits padded with zeros for a FLOAT pin.

15.6.11 Containers: HAL_HideTable HAL_Table State_Sensitive_Table and HAL_HBox

These containers are meant to be used to sensitize (grey out) or hide their children.
Insensitived 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.
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.
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.

* HAL_Hbox is depreceiated - use HAL_Table.
If current panels use it it won't fail. You just won't find it in the GLADE editor anymore.
Future vesions 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.

15.6.12 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
: a String defining ON color of LED. May be any valid gtk.gdk.Color name. Not working on Ubuntu 8.04.

off_color
: String defining OFF color of LED. May be any valid gtk.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 may be represented as `#RRRRGGGGBBBB` strings. These are optional properties which have precedence over `on_color` and `off_color`.

led_size
: LED radius (for square - half of LED's side)

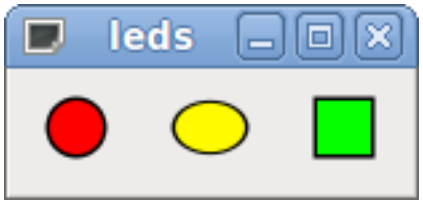
led_shape
: LED Shape. Valid values are 0 for round, 1 for oval and 2 for square shapes.

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 making of HAL pin to control 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.

Example LEDs: 

.

15.6.13 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

It has the following properties:

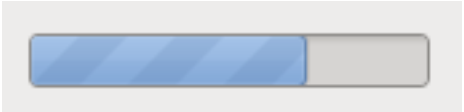
scale
value scale. set maximum absolute value of input. Same as setting the <widgetname>.scale pin. A float, range from -2^{24} to $+2^{24}$.

green_limit
green zone limit lower limit

yellow_limit
yellow zone limit lower limit

red_limit
red zone limit lower limit

text_template
Text template to display the current value of the `<widgetname>` pin. Python formatting may be used for dict `{"value":value}`

Example HAL_ProgressBar:  .

15.6.14 ComboBox

HAL_ComboBox is derived from gtk.ComboBox. It enables choice of a value from a dropdown list.

It exports two HAL pins:

<widgetname>-f
the current value, type FLOAT

<widgetname>-s
the current value, type S32

It has the following property which can be set in Glade:

column
the column index, type S32, defaults to -1, range from -1..100 .

In default mode this widgets 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_F-rW2cL8.

15.6.15 Bars

HAL Bar and VBar widgets for horizontal and vertical bars representing float values. They have one input FLOAT hal pin. 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 vaue 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

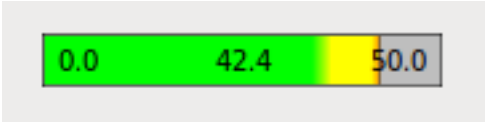
target_width
Width of the line that marks the target value.

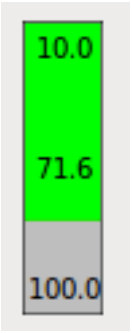
bg_color
Background (inactive) color of bar.

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, values from 0 to 1.

Horizontal bar: 

Vertical bar: 

.

15.6.16 Meter

HAL Meter is a widget similar to PyVCP meter - it represents a float value and 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.

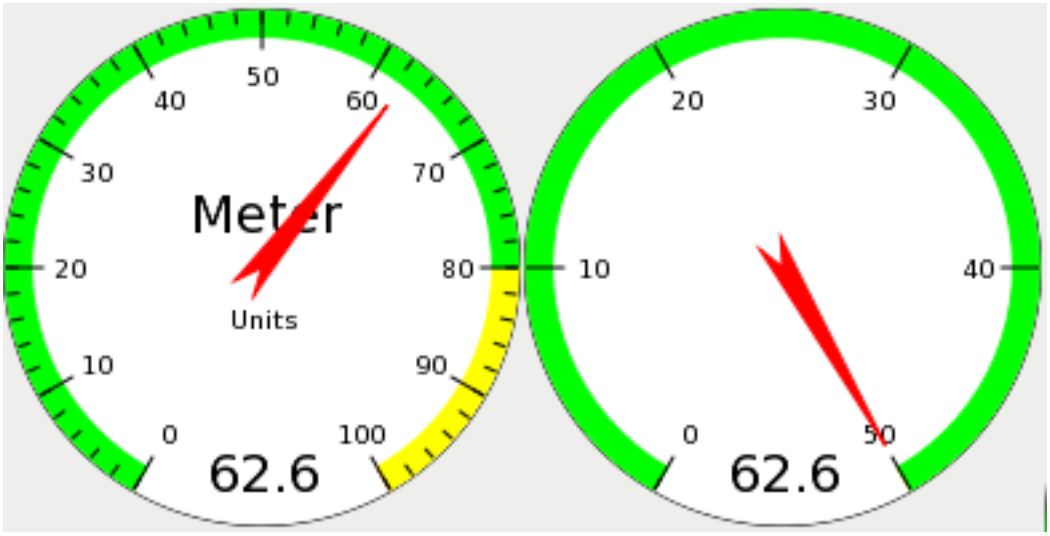
label
Large label above center of meter.

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.

Example HAL Meters: 

. ===
HAL_Graph

This widget is for plotting values over time.

15.6.17 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. It has the following properties:

show tool speed
This displays the tool speed. Defaults true

show commanded
This selects the DRO to use commanded or actual values. Defaults true

use metric units
This selects the DRO to use metric or imperial units. Defaults true

show rapids
This tells the plotter to show the rapid moves. Defaults true

show DTG
This selects the DRO to display the distance-to-go value. Defaults true

show relative
This selects the DRO to show values relative to user system or machine cordinates. Defaults true

show live plot
This tells the plotter to draw or not. Defaults true

show limits
This tells the plotter to show the machine's limits. Defaults 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). Defaults false

show extents
This tells the plotter to show the extents. Defaults true

show tool
This tells the plotter to draw the tool. Defaults true

show program
TODO

use joints mode
Used in non trivialkins machines (eg robots). Defaults false

grid size
Sets the size of the grid. which is only visible in the X, Y and Z view. 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 programically add controls using python and the handler file technique. Defaults to *True*

view
may be any of `x`, `y`, *y2* , `z`, *z2* , `p` (perspective) . Defaults to `z` view.

enable_dro
boolean; whether to draw a DRO on the plot or not. Defaults to `True`

mouse_btn_mode
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

Direct program control

There a couple ways to directly control the widget using Python.

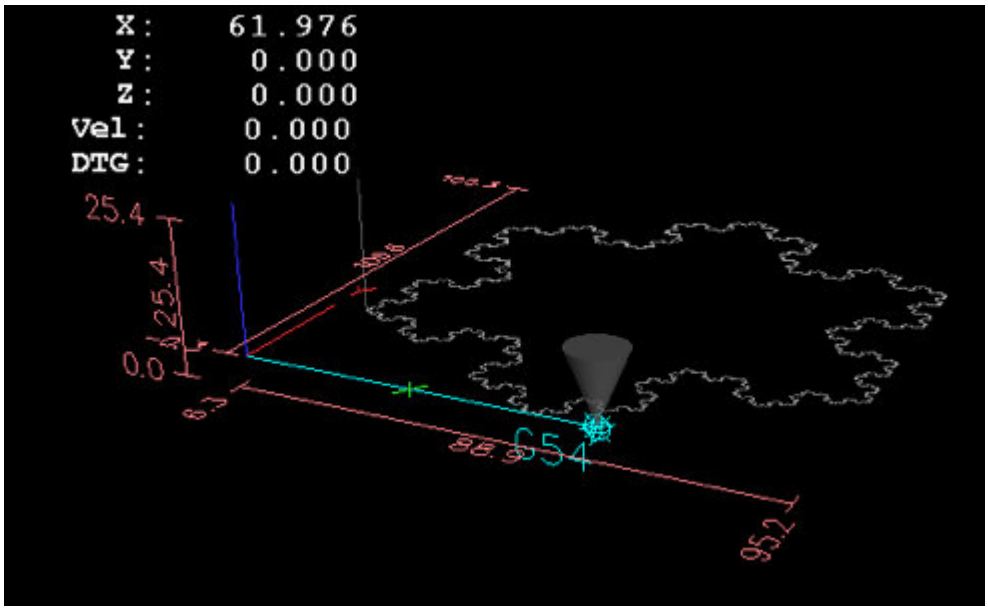
```
Using goobject to set the above listed properties:
    [widget name].set_property('view','P')
    [widget name].set_property('metric_units',False)
    [widget name].set_property('use_default_controls',False)
    [widget name].set_property('enable_dro' False))
    [widget name].set_property('show_program', 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

Example:



15.6.18 HAL_Offset

The HAL_Offset widget is used to display the offset of a single axis. It 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
```

15.6.19 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), 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 backgroud. 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.

Direct program control
There 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()
```

15.6.20 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.

It 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 gcode being G20 or G21

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

Direct program control

Using goobject 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)

[widget name].set_auto_units(state)
    if True the DRO will change units according to active gcode (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, specially needed for ↩
       lathes
    the DRO will display the axis value multiplied by 2
    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"]
            ["DTG", "Rel", "Abs"]
            ["Abs", "DTG", "Rel"]
    Default = ["Rel", "Abs", "DTG"]

[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(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, witch may be of ineterst 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
```

Example, Three Combi_DRO in a window
X = Relative Mode
Y = Absolute Mode
Z = DTG Mode

15.6.21 IconView (File selection) widget

This is touch screen friendly widget to select a file and to change directories.

The 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 directory "jump to" directory, witch 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)
3 = FILEFIRST (show the files first, then the folders),
Default = 2 = FOLDERFIRST

Direct program control

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 gmoccapy
    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 an directory has been  ↩
       selected

[widget name].refresh_filelist()
    Refreshes the filelist, needed if you add a file without changing the  ↩
       directory
```

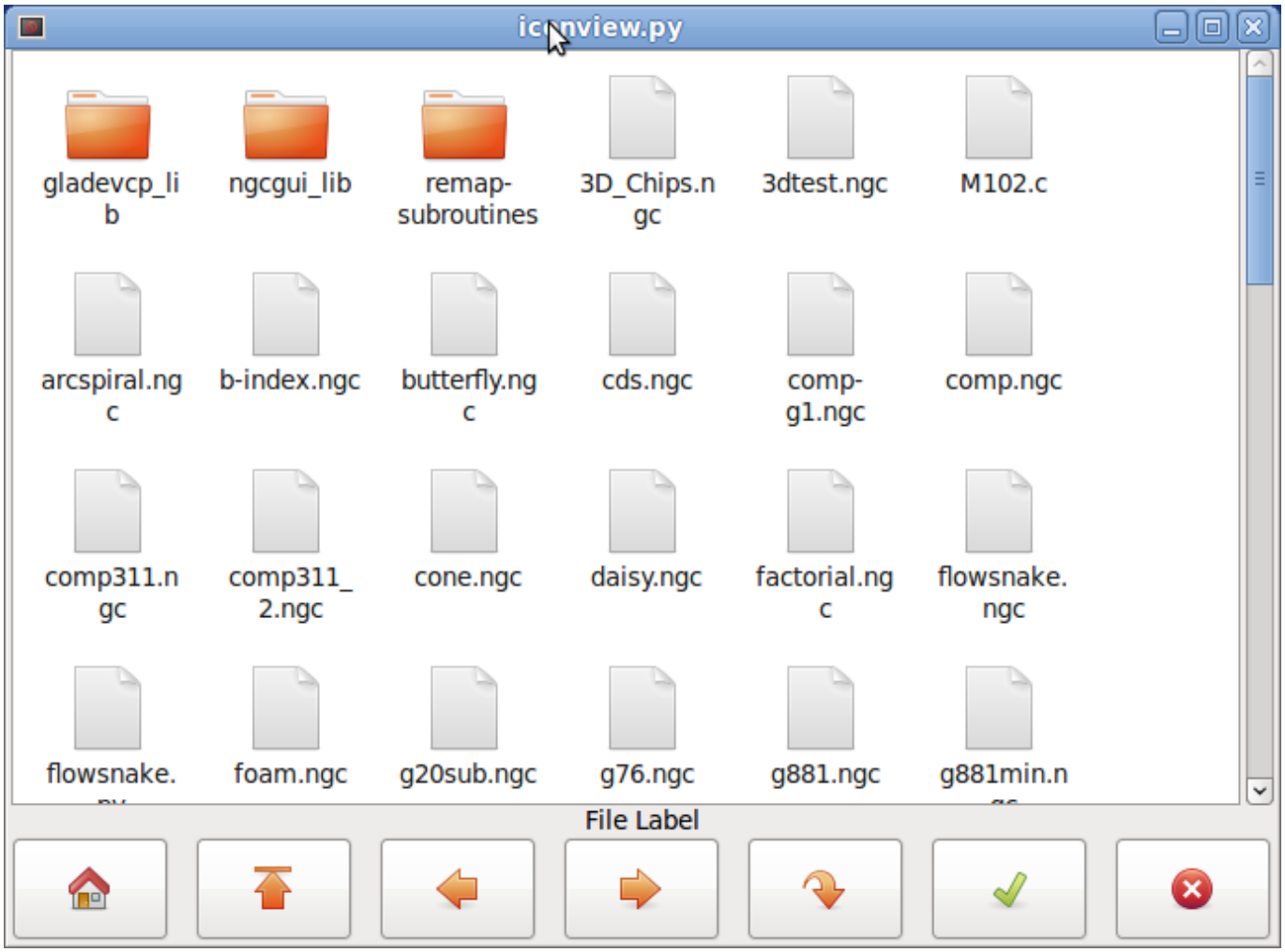
If the button box has been hidden, you can reach the functions of this button through it's clicked signals like so:

```
[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 an directory has been  ↩
       selected
exit
    This signal is Emmit, when the exit button has been pressed to close the  ↩
       IconView
    mostly needed if the application is started as stand alone.
```

Example:



15.6.22 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.
It 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.

Direct program control
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.
```

15.6.23 Tooleditor widget

This is a tooleditor widget for displaying and modifying a tool editor file.
It checks the current file once a second to see if linuxcnc updated it.
It has the following properties:

Hidden Columns

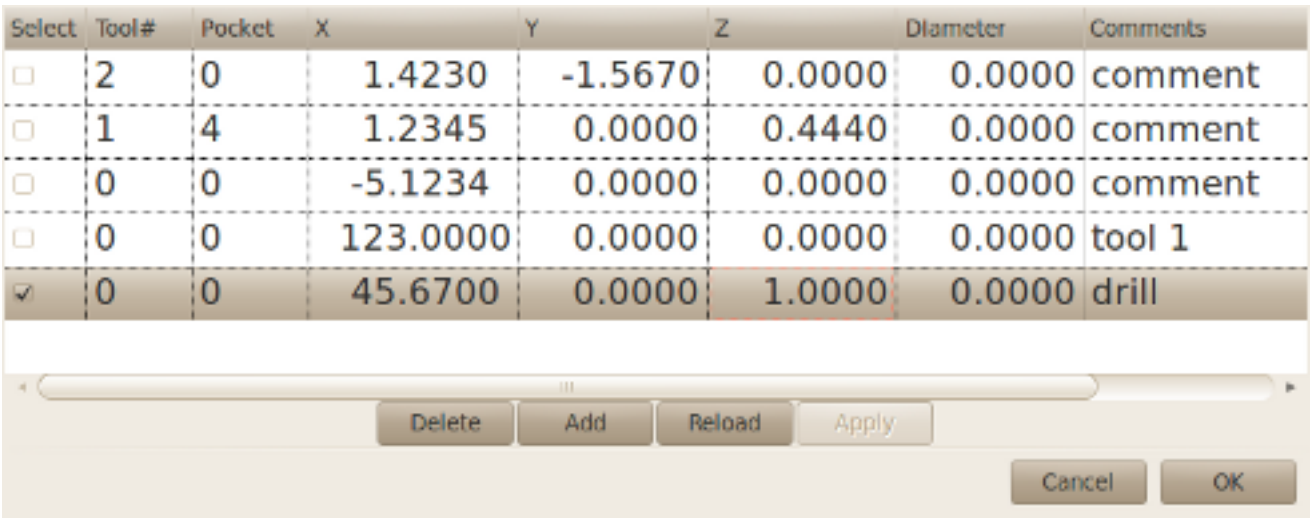
This will hide the given columns: The columns are designated (in order) as such:
s,t,p,x,y,z,a,b,c,u,v,w,d,i,j,q,;
You can hide any number of columns including the select and comments

Direct program control

There 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 uvwij 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
```



15.6.24 Offsetpage

The Offsetpage widget is used to display/edit the offsets of all the axes.
It has convience 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 high light 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.

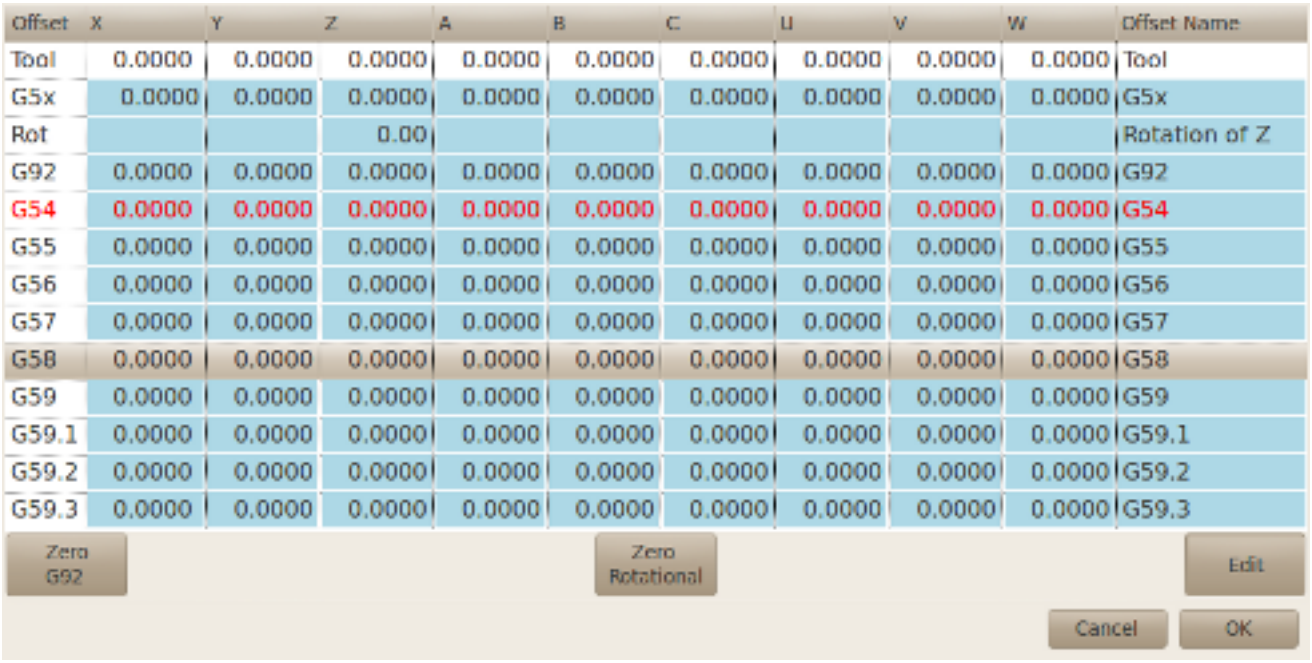
Text template for imperial units
You can use python formatting to display the position with different precision.

Direct program control
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",gtk.gdk.Color('blue'))
[widget name].set_property("foreground_color",gtk.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.
    (eg in case you wish to use your own navigation controls. See 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.
```



15.6.25 HAL_sourceview widget

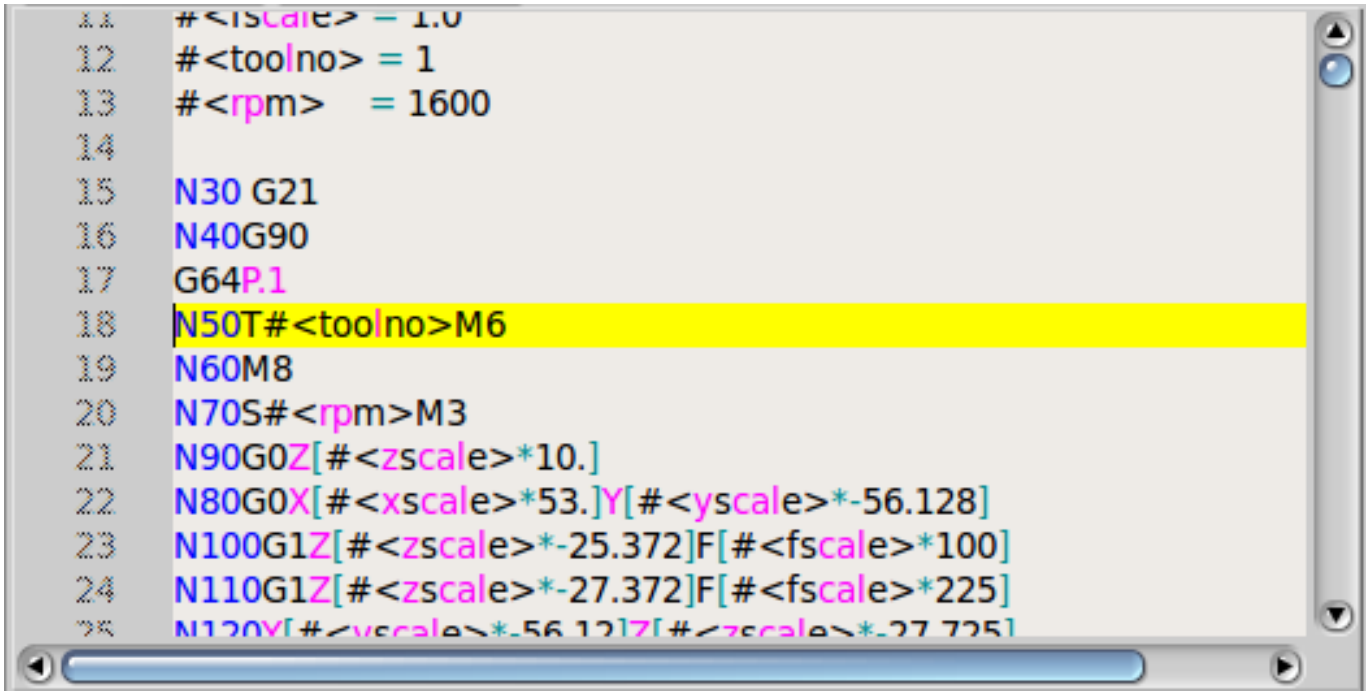
This is for displaying and simple editing of Gcode.
It looks for .ngc highlight specs in ~/share/gtksourceview-2.0/language-specs/ The current running line will be highlighted.
With external python glue code:
*It can search for text, undo and redo changes.
*It can be used for program line selection.

Direct program control

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 back, +
    Searches with mixed case (mixed_case = True) or exact match
[widget name].set_line_number(linenumber)
    Sets the line to high light. Uses the sourceview line numbers.
[widget name].get_line_number()
    returns the currenly high lighted line.
[widget name].line_up()
    Moves the High lighted line up one line
[widget name].line_down()
```

```
    Moves the High lighted 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()
```



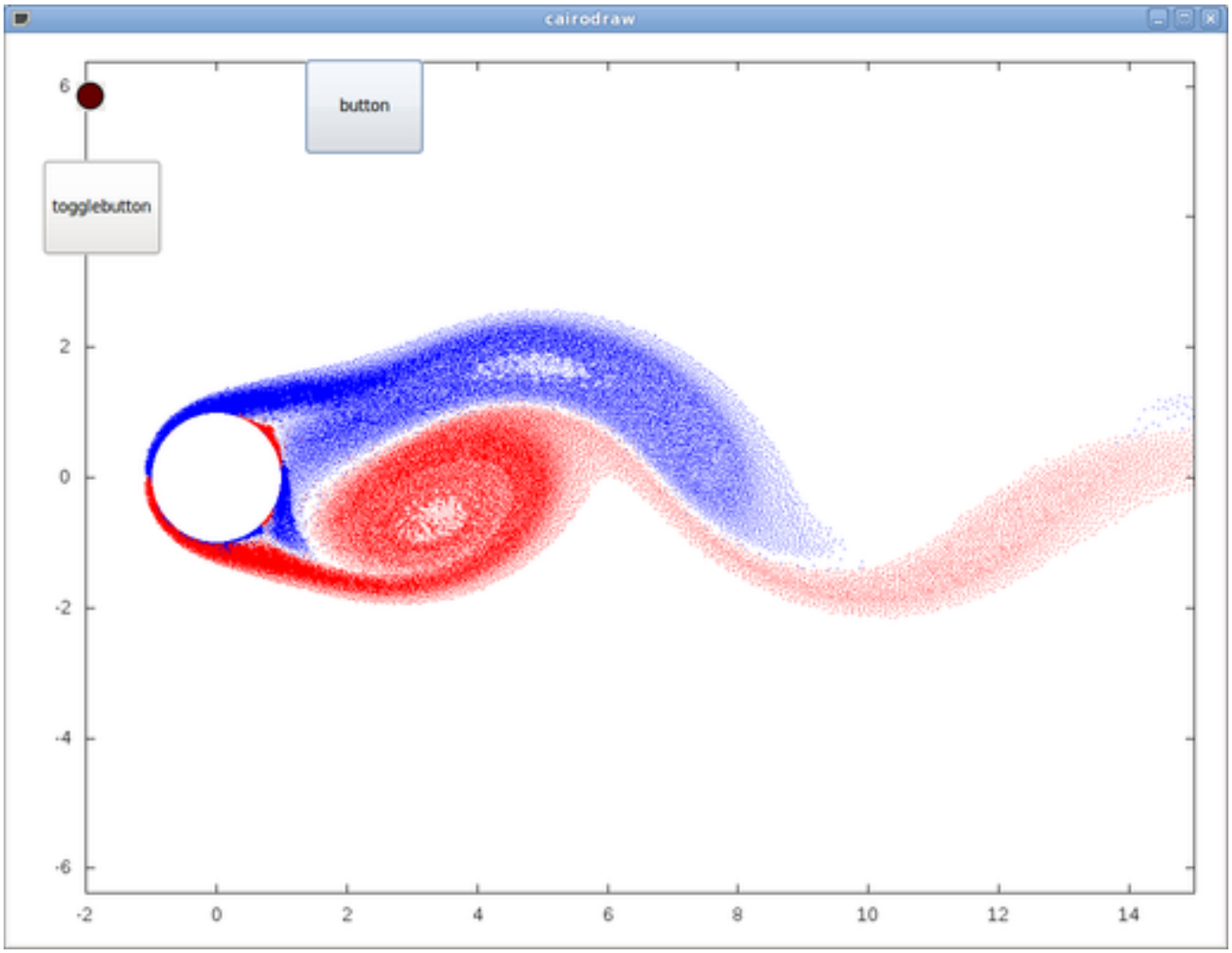
15.6.26 MDI history

This is for displaying and entering MDI codes.
It will automatically grey out when MDI is not available.
Eg during Estop and program running.

15.6.27 Animated function diagrams: HAL widgets in a bitmap

For some applications it might be desirable to have background image - like a functional diagram - and position widgets at appropriate places in that diagram. 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`:



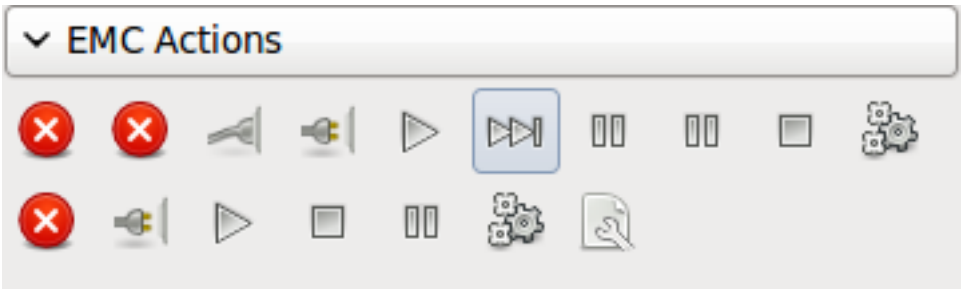
15.7 Action Widgets reference

GladeVcp includes a collection of "canned actions" called EMC Action Widgets for the Glade user interface editor. Other than HAL widgets, which interact with HAL pins, EMC Actions interact with LinuxCNC and the G-code interpreter.

EMC Action Widgets are derived from the Gtk.Action widget. The Action widget in a nutshell:

- it is an object available in Glade
- it 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..)
- it provides an easy way to execute commands without resorting to Python programming.

The appearance of EMC Actions in Glade is roughly as follows:



Tooltip hovers provide a description.

15.7.1 EMC Action widgets

EMC 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.

15.7.2 EMC 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.

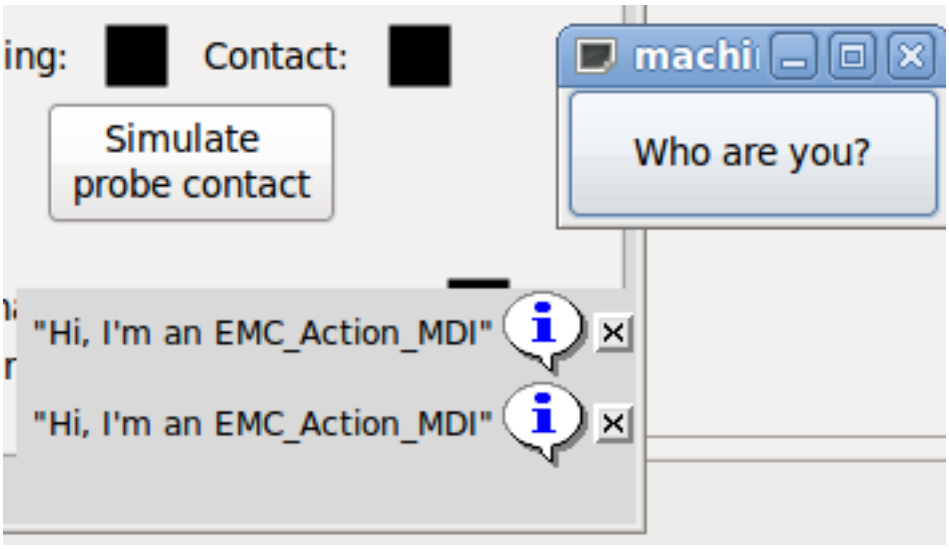
15.7.3 The Action_MDI Toggle and Action_MDI widgets

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.

15.7.4 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:

Open it in Glade and study how it's done. Start Axis, and then start this from a terminal window with `gladevcp whoareyou.ui`. See the `hal_action_mdi1` Action and it's `MDI command` property - this just executes `(MSG, "Hi, I'm an EMC_Action_MDI")` so there should be a message popup in Axis like so:



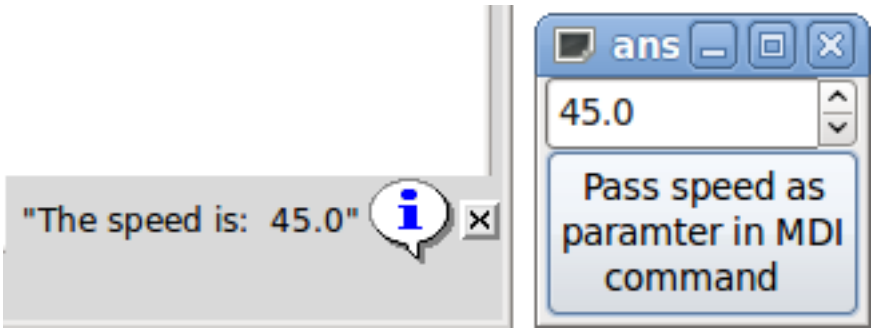
You'll notice that the button associated with the Action_MDI action is grayed out if the machine is off, in E-Stop or the interpreter is running. It will automatically become active when the machine is turned on and out of E-Stop, and the program is idle.

15.7.5 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.

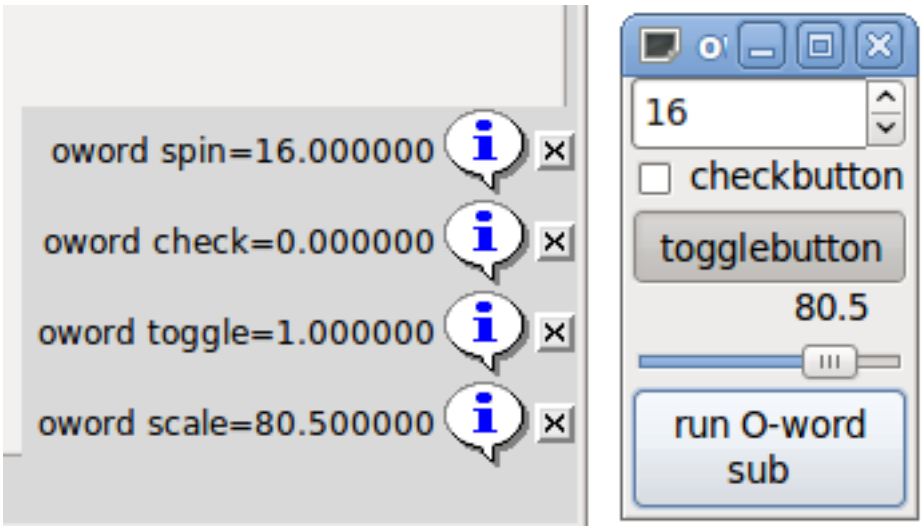
The example UI file is `configs/apps/gladevcp/mdi-command-example/speed.ui`. Here's what you get when running it:



15.7.6 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/gladevcp/mdi-command-example/owordsub.ui`.

Place `nc_files/gladevcp_lib/oword.ngc` so Axis can find it, and run `gladevcp owordsub.ui` from a terminal window. This looks like so:



15.7.7 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 EMC_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 are available which make it saving state before a subroutine call, and restoring state on return much easier.

15.7.8 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: emitted when E-Stop condition occurs, is reset, machine is turned on, or is turned off
  - `state-estop`
  - `state-estop-reset`
  - `state-on`,
  - `state-off`
- mode-related: emitted when LinuxCNC enters that particular mode
  - `mode-manual`
  - `mode-mdi`
  - `mode-auto`
- 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`

15.8 GladeVCP Programming

15.8.1 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.

15.8.2 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!

15.8.3 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 builtin 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.

15.8.4 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.

15.8.4.1 The simple handler model

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.

15.8.4.2 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.

15.8.4.3 The get_handlers protocol

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.

15.8.5 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_autoconnect()` (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 its 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 halfile has been run, and GladeVCP is about to enter the Gtk main loop.

15.8.6 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()`.

15.8.7 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 gladevcp -U option, for example:

```
gladevcp -U debug=42 -U "print 'debug=%d' % debug" ...
```

This should set debug to 2 and confirm that your module actually did it.

15.8.8 Persistent variables in GladeVCP

A 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.

15.8.8.1 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.

15.8.9 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 `gladevcp.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 IniFil methods:

ini.save_state(obj)
saves objs's attributes as per IniFil.vars dictionary and the widget state as described in IniFile.widgets in self.defaults

ini.create_default_ini()
create a .ini file with default values

ini.restore_state(obj)
restore HAL out pins and obj's attributes as saved/initialized to default as above

15.8.10 Saving the state on Gladvcp 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.

15.8.11 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 (actuall 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)
```

15.8.12 Hand-editing .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.

15.8.13 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())
```

15.8.14 Adding timers

Since GladeVCP uses Gtk widgets which rely on the GObject base class, the full glib functionally 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)
```

15.8.15 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 initialisation 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 causese 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)]
```

15.8.16 Examples, and rolling your own GladeVCP application

Visit `linuxcnc_root_directory/configs/apps/gladevcp` for running examples and starters for your own projects.

15.9 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 checkbutton to let it blink with 100msec 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 doesnt 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 isnt 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 doesnt 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 mentione 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}`.

15.10 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/dist-packages/_hal.so` (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.

15.11 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/xembed.py`.

15.12 Adding Custom Widgets

The LinuxCNC Wiki has information on adding custom widgets to GladeVCP. GladeVCP Custom Widgets

Chapter 16

HAL User Interface

16.1 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 (mini, Axis, 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.

```
HALUI = halui
```

An alternate way to invoke it is to include the following in your .hal file. Make sure you use the actual path to your ini file.

```
loadusr halui -ini /path/to/inifile.ini
```

16.2 Halui pin reference

ABORT

- *halui.abort* (bit, in) - pin to send an abort message (clears out most errors)

AXIS

- *halui.axis.n.pos-commanded* (float, out) - Commanded axis position in machine coordinates
- *halui.axis.n.pos-feedback* (float, out) - Feedback axis position in machine coordinates
- *halui.axis.n.pos-relative* (float, out) - Commanded axis position in relative coordinates

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

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.direct-value* (bit, in) - false when using encoder to change counts, true when setting counts directly. The *count-enable* pin must be true.
- *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

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

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

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.

Jog <n> is a number between 0 and 8 and *selected*.

- *halui.jog-deadband* (float, in) - deadband for analog jogging (smaller jogging speed requests are not performed)
- *halui.jog-speed* (float, in) - pin for setting jog speed for minus/plus jogging
- *halui.jog.<n>.analog* (float, in) - analog velocity input for jogging (useful with joysticks or other analog devices)
- *halui.jog.<n>.increment* (float,in) - pin for setting the jog increment for axis <n> when using increment-minus or increment-plus to jog.
- *halui.jog.<n>.increment-minus* (bit, in) - pin for moving the <n> axis one increment in the minus direction for each off to on transition.
- *halui.jog.<n>.increment-plus* (bit, in) - pin for moving the <n> axis one increment in the plus direction for each off to on transition.
- *halui.jog.<n>.minus* (bit, in) - pin for jogging axis <n> in negative direction at the halui.jog.speed velocity
- *halui.jog.<n>.plus* (bit, in) - pin for jogging axis <n> in positive direction at the halui.jog.speed velocity
- *halui.jog.selected.increment* (float,in) - pin for setting the jog increment for the selected axis when using increment-minus or incremet-plus to jog.
- *halui.jog.selected.increment-minus* (bit, in) - pin for moving the selected axis one increment in the minus direction for each off to on transition.
- *halui.jog.selected.increment-plus* (bit, in) - pin for moving the selected axis one increment in the plus direction for each off to on transition.

- *halui.jog.selected.minus* (bit, in) - pin for jogging the selected axis in negative direction at the halui.jog.speed velocity
- *halui.jog.selected.plus* (bit, in) - pin for jogging the selected axis in positive direction at the halui.jog.speed velocity

Joint <n> is a number between 0 and 8 and *selected*.

- *halui.joint.<n>.has-fault* (bit, out) - status pin telling the joint has a fault
- *halui.joint.<n>.home* (bit, in) - pin for homing the specific joint
- *halui.joint.<n>.is-homed* (bit, out) - status pin telling that the joint is homed
- *halui.joint.<n>.is-selected bit* (bit, out) - status pin a joint is selected* internal halui
- *halui.joint.<n>.on-hard-max-limit* (bit, out) - status pin telling joint <n> is on the positive hardware limit switch
- *halui.joint.<n>.on-hard-min-limit* (bit, out) - status pin telling joint <n> is on the negative hardware limit switch
- *halui.joint.<n>.on-soft-max-limit* (bit, out) - status pin telling joint <n> is at the positive software limit
- *halui.joint.<n>.on-soft-min-limit* (bit, out) - status pin telling joint <n> is at the negative software limit
- *halui.joint.<n>.select* (bit, in) - select joint (0..8) - internal halui
- *halui.joint.<n>.unhome* (bit, in) - unhomes this joint
- *halui.joint.selected* (u32, out) - selected joint (0..8) - internal halui
- *halui.joint.selected.has-fault* (bit, out) - status pin telling that the joint <n> has a fault
- *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.unhome* (bit, in) - pin for unhoming the selected joint.

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

MACHINE

- *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

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. The *count-enable* pin must be true.
- *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.

MDI

Sometimes the user wants to add more complicated tasks to be performed by the activation of a HAL pin. This is possible using the following MDI commands scheme:

- The MDI_COMMAND is added to the ini file in the [HALUI] section.

```
[HALUI]
MDI_COMMAND = G0 X0
```

- 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.
- *halui.mdi-command-<nn>* (bit, in) - halui will try to send the MDI command defined in the ini. This will not always succeed, depending on the operating mode LinuxCNC is in (e.g. while in AUTO halui can't successfully send MDI commands). If the command succeeds then it will place LinuxCNC in the MDI mode and then back to Manual mode.

JOINT SELECTION

- *halui.joint.select* (u32, in) - select joint (0..8) - internal halui
- *halui.joint.selected* (u32, out) - joint (0..8) selected* internal halui
- *halui.joint.x.select bit* (bit, in) - pins for selecting a joint* internal halui
- *halui.joint.x.is-selected bit* (bit, out) - indicates joint selected* internal halui

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

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

SPINDLE OVERRIDE

- *halui.spindle-override.count-enable* (bit, in) - must be true for *counts* or *direct-value* to work.
- *halui.spindle-override.counts* (s32, in) - counts * scale = SO percentage
- *halui.spindle-override.decrease* (bit, in) - pin for decreasing the SO (-=scale)
- *halui.spindle-override.direct-value* (bit, in) - false when using encoder to change counts, true when setting counts directly. The *count-enable* pin must be true.
- *halui.spindle-override.increase* (bit, in) - pin for increasing the SO (+=scale)
- *halui.spindle-override.scale* (float, in) - pin for setting the scale on changing the SO
- *halui.spindle-override.value* (float, out) - current SO value

SPINDLE

- *halui.spindle.brake-is-on* (bit, out) - indicates brake is on
- *halui.spindle.brake-off* (bit, in) - pin for deactivating spindle/brake
- *halui.spindle.brake-on* (bit, in) - pin for activating spindle-brake
- *halui.spindle.decrease* (bit, in) - decreases spindle speed
- *halui.spindle.forward* (bit, in) - starts the spindle with CW motion
- *halui.spindle.increase* (bit, in)- increases spindle speed
- *halui.spindle.is-on* (bit, out) - indicates spindle is on (either direction)
- *halui.spindle.reverse* (bit, in)- starts the spindle with a CCW motion
- *halui.spindle.runs-backward* (bit, out) - indicates spindle is on, and in reverse

- *halui.spindle.runs-forward* (bit, out) - indicates spindle is on, and in forward
- *halui.spindle.start* (bit, in) - starts the spindle
- *halui.spindle.stop* (bit, in) - stops the spindle

TOOL

- *halui.tool.length-offset* (float, out) - indicates current applied tool-length-offset
- *halui.tool.number* (u32, out) - indicates current selected tool

Chapter 17

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
```

17.1 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.

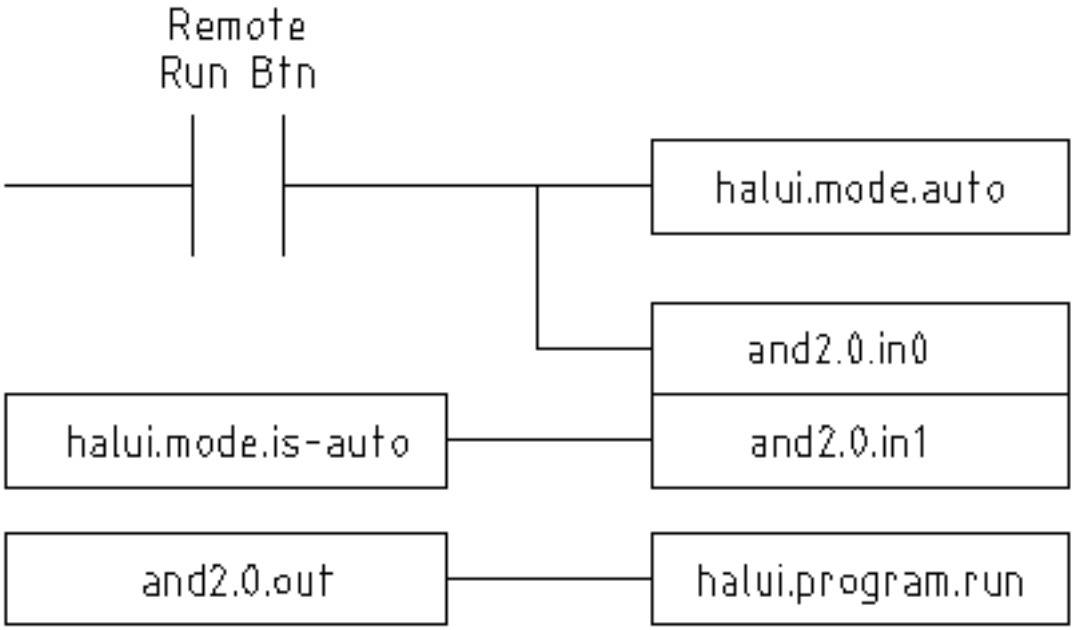


Figure 17.1: Remote Start Example

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
```

17.2 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 gcode program, when the external controller is ready it will make its output true. This will signal LinuxCNC that it should resume executing Gcode.

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.

Part IV

Hardware Drivers

Chapter 18

Parallel Port Driver

18.1 Parport

Parport 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 output, a port provides a total of 12 outputs and 5 inputs. If configured as input, 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, the *x* mode allows a more flexible mode with 8 outputs, and 9 inputs. In other 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 machines, 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 Block Diagram shows two block diagrams, one showing the driver when the data group is configured for output, and one showing it configured for input. For *x* mode, refer to the pin listing of *halcmd show pin* for pin direction assignment.

The parport driver can control up to 8 ports (defined by MAX_PORTS in hal_parport.c). The ports are numbered starting at zero.

18.1.1 Installing

```
loadrt hal_parport cfg="<config-string>"
```

Using the Port Index I/O addresses below 16 are treated as port indexes. This is the simplest way to install the parport driver and cooperates with the Linux parport_pc driver if it is loaded. This will use the address Linux has detected for parport 0.

```
loadrt hal_parport cfg="0"
```

Using the Port Address The configure string consists of a hex port address, followed by an optional direction, repeated for each port. The direction is *in*, *out*, or *x* and determines the direction of the physical pins 2 through 9, and whether to create input HAL pins for the physical control pins. If the direction is not specified, the data group defaults to output. For example:

```
loadrt hal_parport cfg="0x278 0x378 in 0x20A0 out"
```

This example installs drivers for one port at 0x0278, with pins 2-9 as outputs (by default, since neither *in* nor *out* was specified), one at 0x0378, with pins 2-9 as inputs, and one at 0x20A0, with pins 2-9 explicitly specified as outputs. Note that you must know the base address of the parallel port to properly configure the driver. For ISA bus ports, this is usually not a problem, since the port is almost always at a *well known* address, like 0278 or 0378 which is typically configured in the system BIOS. The address for a PCI card is usually shown in *lspci -v* in an *I/O ports* line, or in the kernel message log after executing *sudo modprobe -a parport_pc*. There is no default address; if <*config-string*> does not contain at least one address, it is an error.

Port Address

For those who build their own hardware, one safeguard against shorting out an on-board parallel port - or even the whole motherboard - is to use an add-on parallel port card. Even if you don't need the extra layer of safety, a parport card is a good way to add extra I/O lines with LinuxCNC.

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 these 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).

You can then open an editor and put the addresses into the appropriate place in your .hal file.

```
loadrt hal_parport cfg="0x378 0xc000"
```

You must also direct LinuxCNC to run the *read* and *write* functions for the second card. For example,

```
addf parport.1.read base-thread 1
addf parport.1.write base-thread -1
```

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.

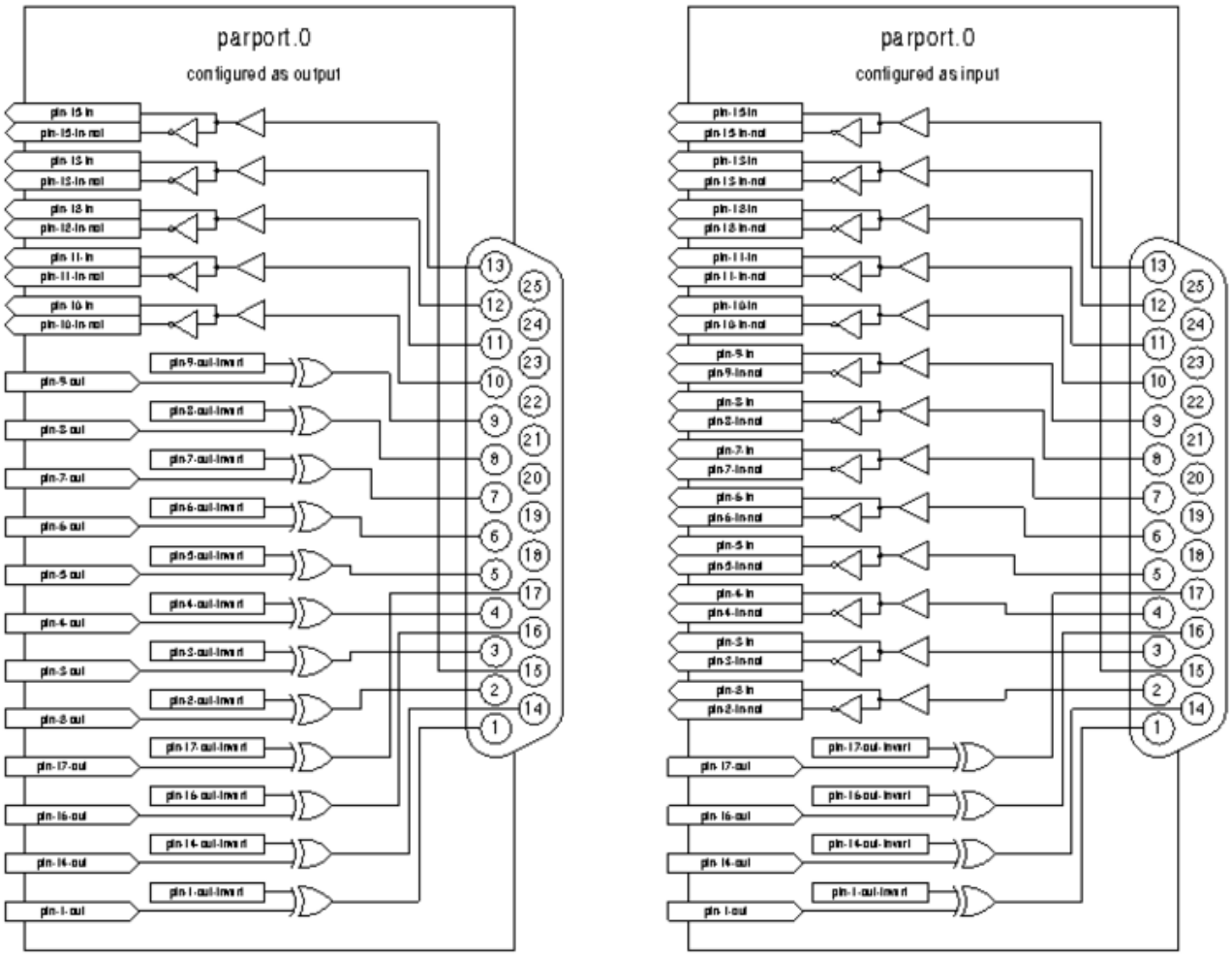


Figure 18.1: Parport Block Diagram

18.1.2 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*.

Pins 2 through 9 are part of the data group and are output pins if the port is defined as an output port. (Output is the default.) Pins 1, 14, 16, and 17 are outputs in all modes. These HAL pins control the state of the corresponding physical pins.

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*.

Pins 10, 11, 12, 13, and 15 are always input pins. Pins 2 through 9 are input pins only if the port is defined as an input port. 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 — 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. In *x* mode, pins 1, 14, 16, and 17 are also input pins.

18.1.3 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.

18.1.4 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 *-out-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.

18.1.5 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.

18.1.6 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 specificed by parport.n.reset-time.

For example:

[1] In the LinuxCNC packages for Ubuntu, the file /etc/modprobe.d/emc2 generally prevents *parport_pc* from being automatically loaded.

```
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.

Chapter 19

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.

19.1 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 IIIOIIOO 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.

19.2 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.

19.3 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.

19.4 Functions

- *(funct) ax5214.<boardnum>.read* — Reads all digital inputs on one board.
- *(funct) ax5214.<boardnum>.write* — Writes all digital outputs on one board.

Chapter 20

GS2 VFD Driver

This is a userspace HAL program for the GS2 series of VFD's at Automation Direct.

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

20.1 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 name of 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.

Note
That if there are serial configuration errors, turning on verbose may result in a flood of timeout errors.

20.2 Pins

Where <n> is gs2_vfd or the name given during loading with the -n option.

- *<n>.DC-bus-volts* (float, out) The 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).

20.3 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)

For an example of using this component to drive a spindle see the GS2 Spindle example.

Chapter 21

Mesa HostMot2 Driver

21.1 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.

21.2 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.

21.3 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.

21.4 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.

21.5 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:

21.5.1 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.

21.5.2 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.

21.6 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.

21.7 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.

21.8 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>/
```

21.9 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.

21.10 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.

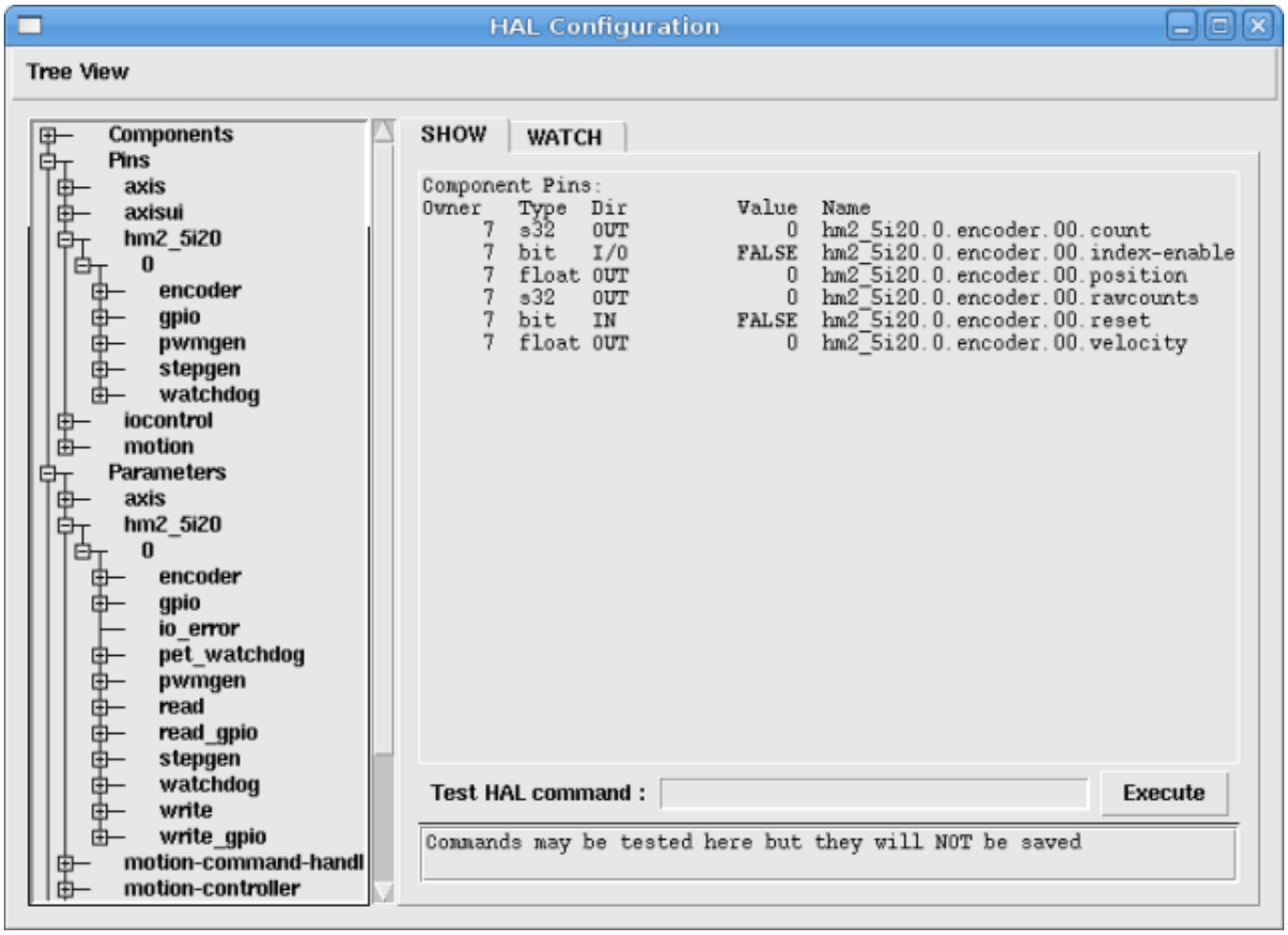


Figure 21.1: 5i20 HAL Pins

21.11 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	12 (+IM)	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	12 (+IM)	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
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 (pwmgen, stepgen, and encoder) by setting the count lower. Any I/O pins belonging to disabled module instances become GPIOs.

21.12 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.

21.12.1 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.

21.12.2 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.

21.13 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.[1]

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:

21.13.1 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.

21.13.2 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. position = counts / position_scale

[1] At present, the firmware supports multi-phase stepper outputs, but the driver doesn't. Interested volunteers are solicited.

- *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.

21.13.3 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.

21.14 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:

21.14.1 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.

21.14.2 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.

21.14.3 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.

21.15 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:

21.15.1 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.

21.15.2 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 revelution 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.

21.16 5i25 Configuration

21.16.1 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.

21.16.2 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"
```

21.16.3 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 usuage.

21.16.4 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)

21.17 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.

Chapter 22

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.

22.1 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, seen 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.

22.2 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.

22.3 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.

Chapter 23

Opto22 Driver

PCI AC5 ADAPTER CARD / HAL DRIVER

23.1 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 thats 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!

23.2 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.

23.3 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.

23.4 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.

23.5 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.

23.6 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 (thats the outputs) then 0 twelve times (thats 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.

000000000000000011111111

16 zeros for the 16 inputs and 8 ones for the 8 outputs

Which converts to FF on the calculator so 0xff is the number to use for portconfig0 and/or portconfig1 when loading the driver.

23.7 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.

Chapter 24

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 busses may be used, and each bus may have up to 8 (or possibly 16 PPMC) devices on it.

24.1 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 frequeency. (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*

24.2 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 axis.#.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 ur 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 ppmc 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.

24.3 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 610Hz, maximum is 500KHz.
- *(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 100ns. Applies to a group of four consecutive step generators, as indicated by <*channel-range*>. Values between 200 ns and 25.5 us can be specified.
- *(USC u32) ppmc.<port>.stepgen.<channel-range>.pulse-width* - Sets width of step pulses, in units of 100ns. Applies to a group of four consecutive step generators, as indicated by <*channel-range*>. Values between 200 ns and 25.5 us may be specified.
- *(USC u32) ppmc.<port>.stepgen.<channel-range>.pulse-space-min* - Sets minimum time between pulses, in units of 100ns. Applies to a group of four consecutive step generators, as indicated by <*channel-range*>. Values between 200 ns and 25.5 us can be specified. The maximum step rate is: $\frac{1}{100\text{ns} * (pulsewidth + pulsespacemin)}$
- *(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.

24.4 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, stepgens, PWMs) on one port. These writes are organized into blocks of contiguous registers to be written in a block to minimize CPU overhead.

Chapter 25

Pluto P Driver

25.1 General Info

The Pluto-P is a FPGA board featuring the ACEX1K chip from Altera.

25.1.1 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.

25.1.2 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.

25.1.3 Physical Pins

- Read the ACEX1K datasheet for information about input and output voltage thresholds. The pins are all configured in *LVTTL/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 (20k-ohms min, 50k-ohms 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 6.5ms 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 25ns. Digital filtering has been added to filter pulses shorter than 175ns (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-ohm 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.

25.1.4 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 xor DOWN0*) or STEPGEN0 (*LED = STEP0 xor DIR0*).

25.1.5 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.

25.1.6 PC interface

- Only a single pluto_servo or pluto_step board is supported.

25.1.7 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.

25.1.8 For more information

Some additional information about it is available from KNJC LLC and from the developer's blog.

25.2 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 40MHz sample rate. The counters operate in *4x* mode. The maximum useful quadrature rate is 8191 counts per LinuxCNC servo cycle, or about 8MHz for LinuxCNC's default 1ms 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 19.5kHz (40MHz / 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 us on machines tested so far, enabling servo rates above 1kHz.

25.2.1 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 ouput 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

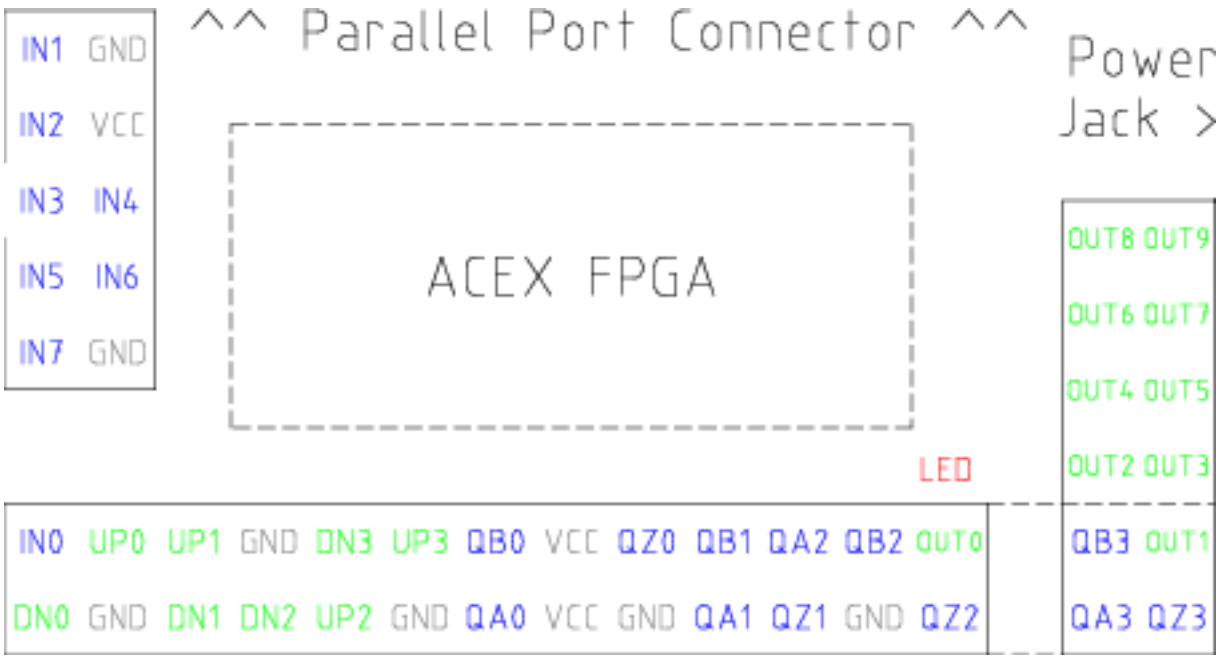


Figure 25.1: Pluto-Servo Pinout

Table 25.1: 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
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

25.2.2 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.

25.2.3 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*.

25.2.4 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.

25.3 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 312.5kHz maximum step rate, programmable step length, space, and direction change times
- 14 dedicated digital outputs
- 16 dedicated digital inputs
- EPP communication with the PC

25.3.1 Pinout

- *STEPx* - The *step* (clock) output of stepgen channel *x*
- *DIRx* - The *direction* output of stepgen channel *x*
- *INx* - Dedicated digital input #x
- *OUTx* - 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.

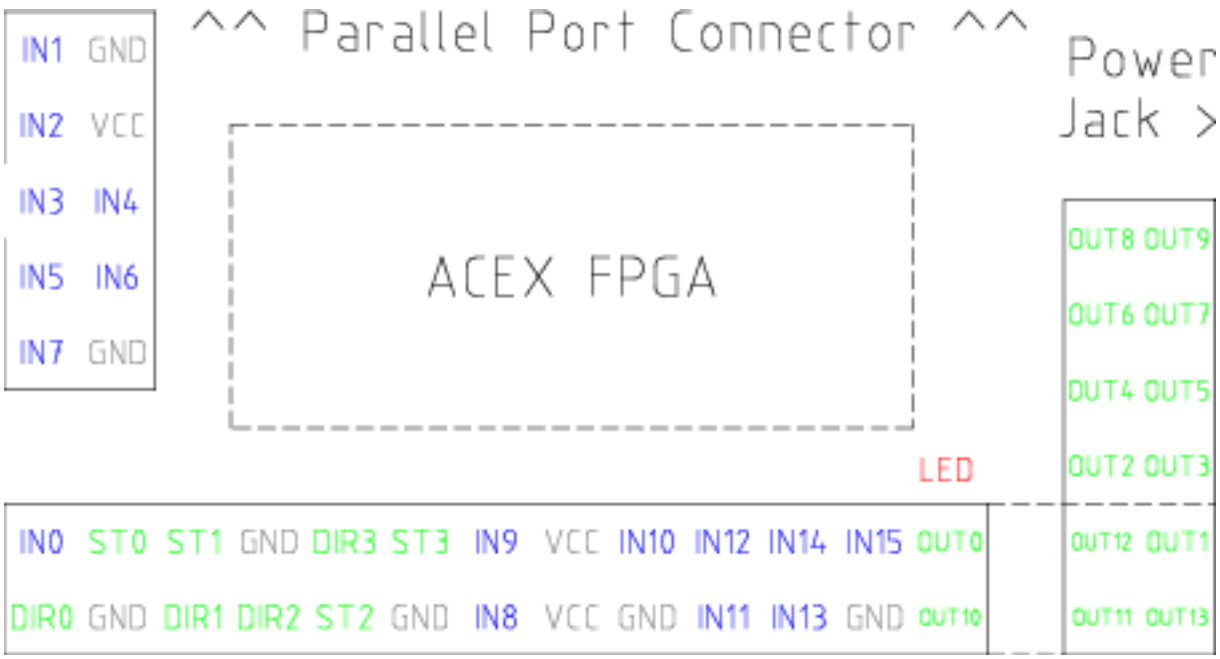


Figure 25.2: Pluto-Step Pinout

25.3.2 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.

25.3.3 Step Waveform Timings

The firmware and driver enforce step length, space, and direction change times. Timings are rounded up to the next multiple of 1.6μs, with a maximum of 49.6μs. 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.

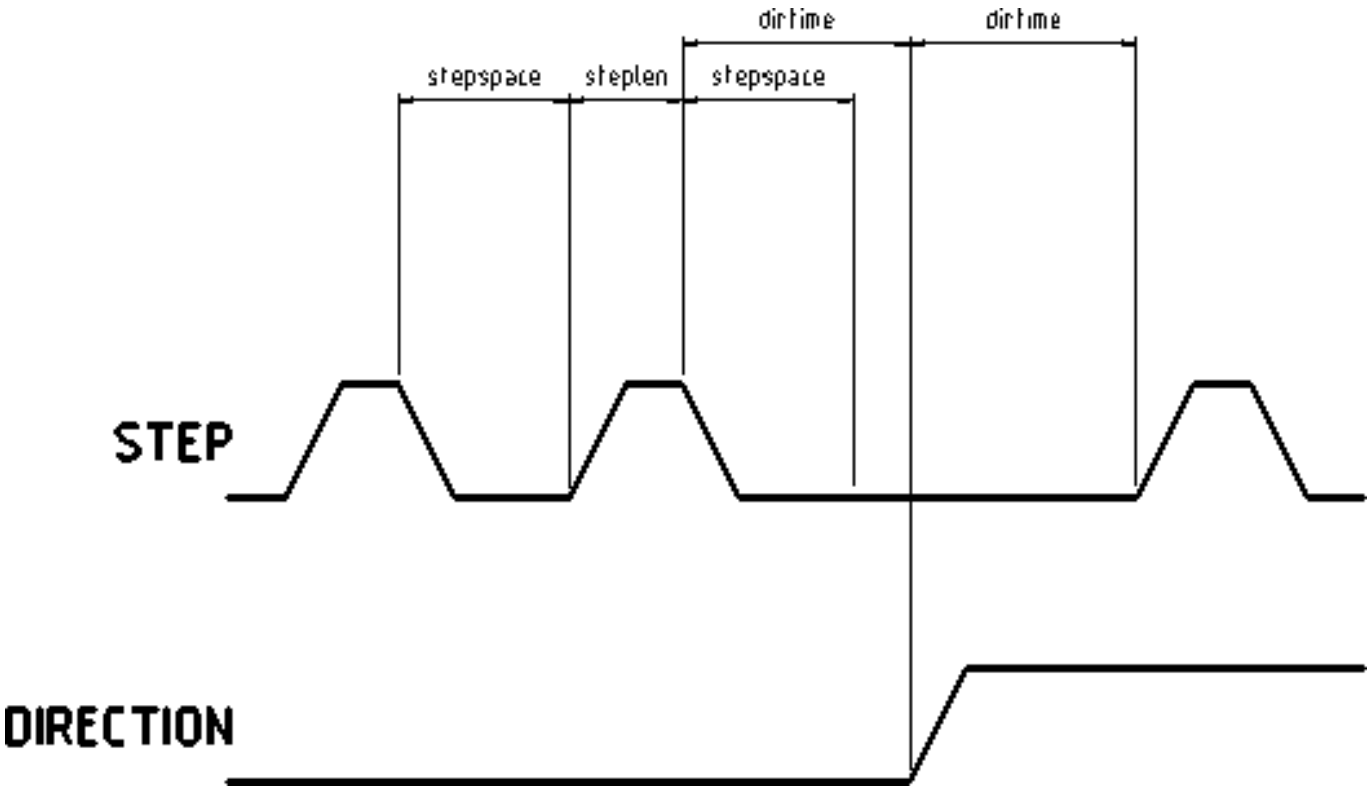


Figure 25.3: Pluto-Step Timings

25.3.4 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*.

Chapter 26

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.

26.1 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.

26.2 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 `IIOO` 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.

26.3 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.

26.4 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.

Chapter 27

ShuttleXpress

27.1 Description

shuttlexpress is a userspace HAL component that interfaces Contour Design's ShuttleXpress device with LinuxCNC's HAL. 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.

If it is started without command-line arguments, it will probe all /dev/hidraw* device files for ShuttleXpress devices, and use all devices found. If it is started with command-line arguments, only will only probe the devices specified.

Warning
The ShuttleXpress device has an internal 8-bit counter for the current jog-wheel position. The shuttlexpress driver can not know this value until the ShuttleXpress 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 ShuttleXpress 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.

27.2 Setup

The shuttlexpress module needs read permission on the /dev/hidraw* device files. This can be accomplished by adding a file /etc/udev/rules.d/99-shuttlexpress.rules, with the following contents:

```
SUBSYSTEM=="hidraw", ATTRS{idVendor}=="0b33", ATTRS{idProduct}=="0020", MODE="0444"
```

27.3 Pins

- *shuttlexpress.<DeviceNumber>.button-<ButtonNumber>* (bit out) The ShuttleXpress has five buttons around the outside of the device, numbered 0 through 4. 0 is the counter-clockwise-most button, 4 is the clockwise-most button. These pins are True (1) when the button is pressed.
- *shuttlexpress.<DeviceNumber>.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.
- *shuttlexpress.<DeviceNumber>.counts* (s32 out) Accumulated counts from the jog wheel (the inner wheel).

- *shuttlexpress.<DeviceNumber>.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.
- *shuttlexpress.<DeviceNumber>.spring-wheel-f* (float out) The current deflection of the spring-wheel (the outer wheel). It's 0 at rest, -1 at the counter-clockwise extreme, and +1 at the clockwise extreme. (The ShuttleXpress device reports the spring-wheel position quantized from -7 to +7, so this pin reports only 15 discrete values in it's range.)

Chapter 28

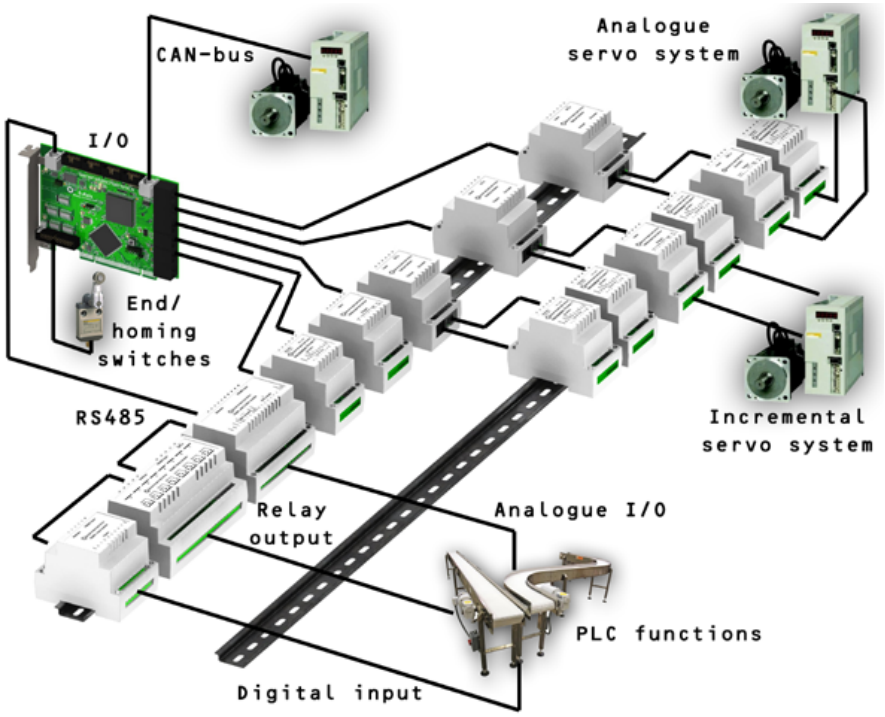
General Mechatronics Driver

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 axis. 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:

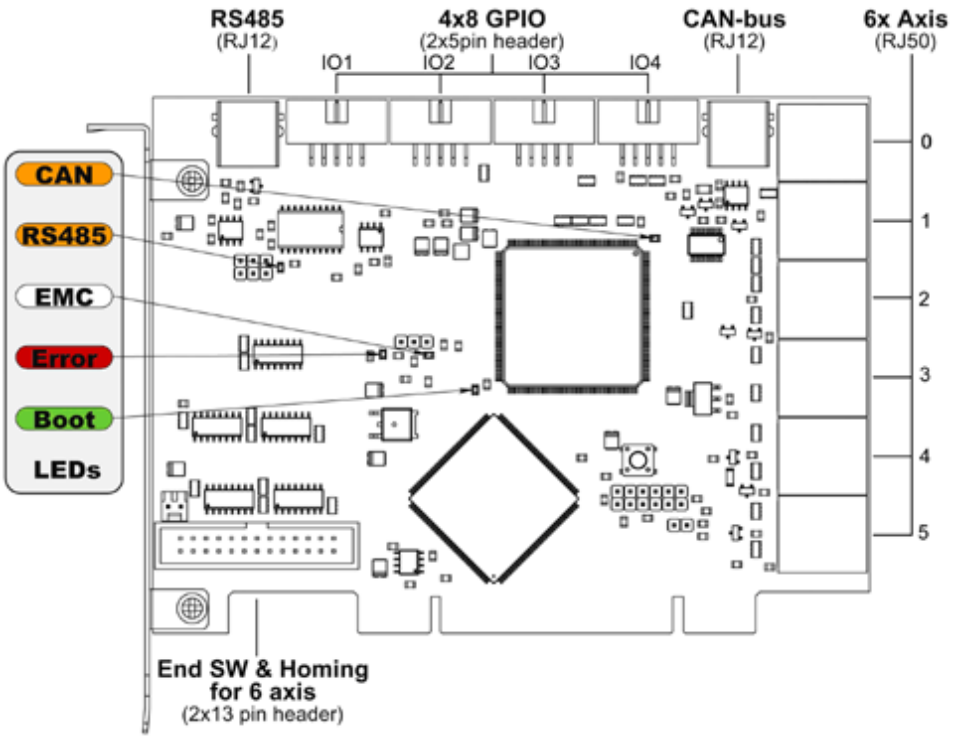


Figure 28.1: GM6-PCI card connectors and LEDs

28.1 I/O connectors

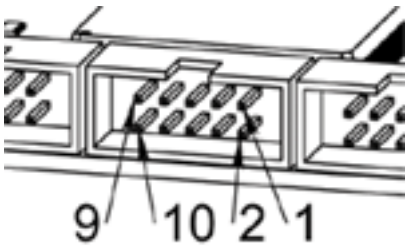


Figure 28.2: Pin numbering of GPIO connectors

Table 28.1: 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 form 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
```

28.1.1 Pins

Table 28.2: 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.

28.1.2 Parameters

Table 28.3: 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.

28.2 Axis connectors

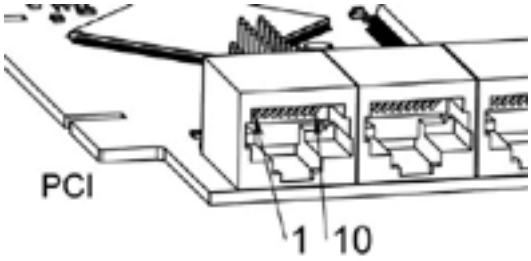


Figure 28.3: Pin numbering of axis connectors

Table 28.4: 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

28.2.1 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:

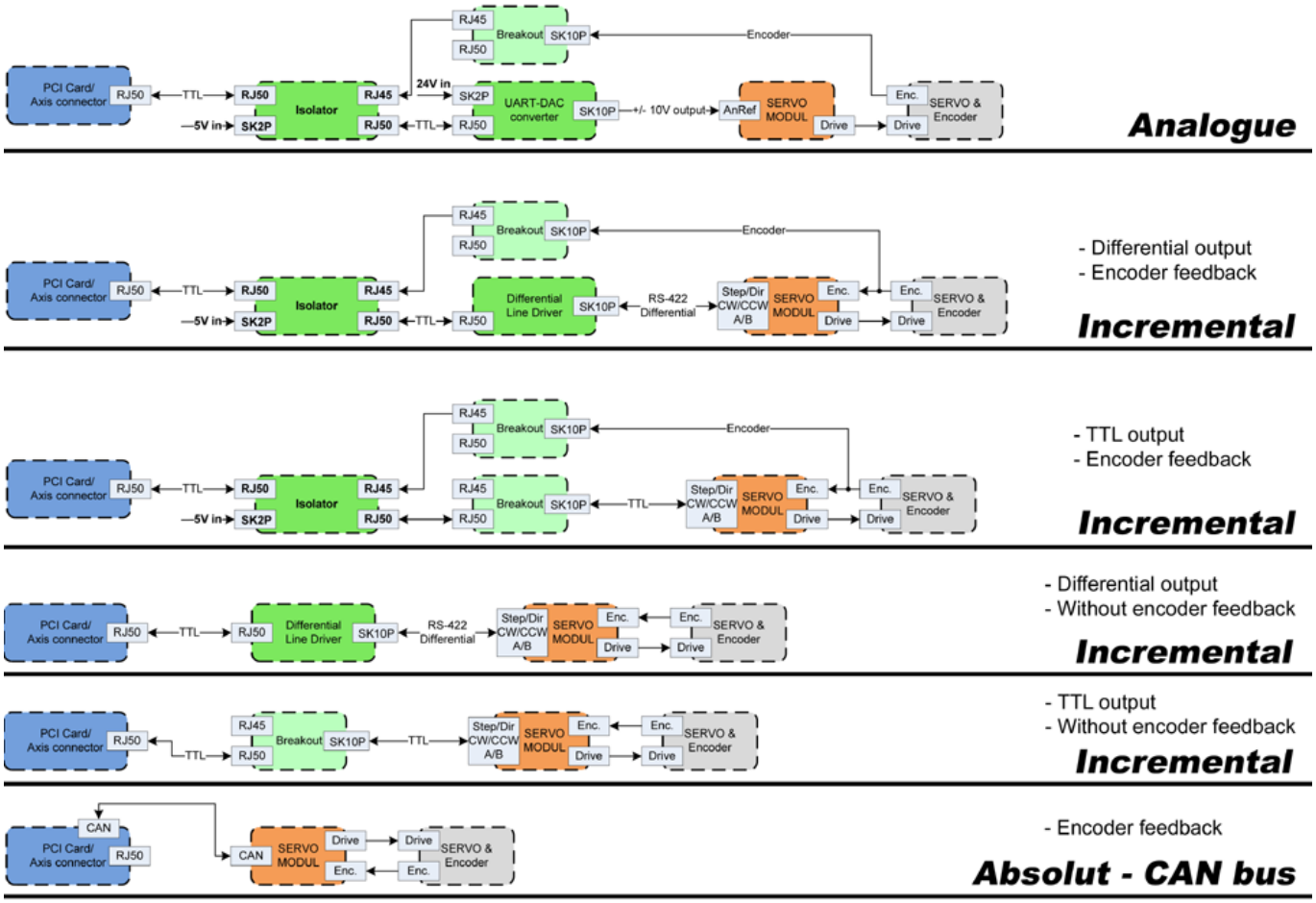


Figure 28.4: Servo axis interfaces

28.2.2 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 form 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
```

28.2.2.1 Pins

Table 28.5: 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-velocity-estimate, the velocity output is 0.

28.2.2.2 Parameters

Table 28.6: 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.
.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.

Table 28.6: (continued)

Parameters	Type and Read/Write	Parameter description
.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.

28.2.2.3 HAL example

Setting encoder module of axis 0 to receive 500 CPR quadrature encoder signal and use reset to round position.

```
setp gm.0.encoder.0.counter-mode 0         # 0: quad, 1: stepDir
setp gm.0.encoder.0.index-mode 1           # 0: reset pos at index, 1:round pos at index
setp gm.0.encoder.0.counts-per-rev 2000      # GM process encoder in 4x mode, 4x500=2000
setp gm.0.encoder.0.index-invert 0
setp gm.0.encoder.0.min-speed-estimate 0.1 # in position unit/s
setp gm.0.encoder.0.position-scale 20000   # 10 encoder rev cause the machine to
                                             move one position unit (10x2000)
```

Connect encoder position to LinuxCNC position feedback:

```
net Xpos-fb gm.0.encoder.0.position => axis.0.motor-pos-fb
```

28.2.3 Stepgen module

The GM6-PCI motion control card has six stepgen modules, one for each axis. 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 form 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
```

28.2.3.1 Pins

Table 28.7: 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.

28.2.3.2 Parameters

Table 28.8: 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.
.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:

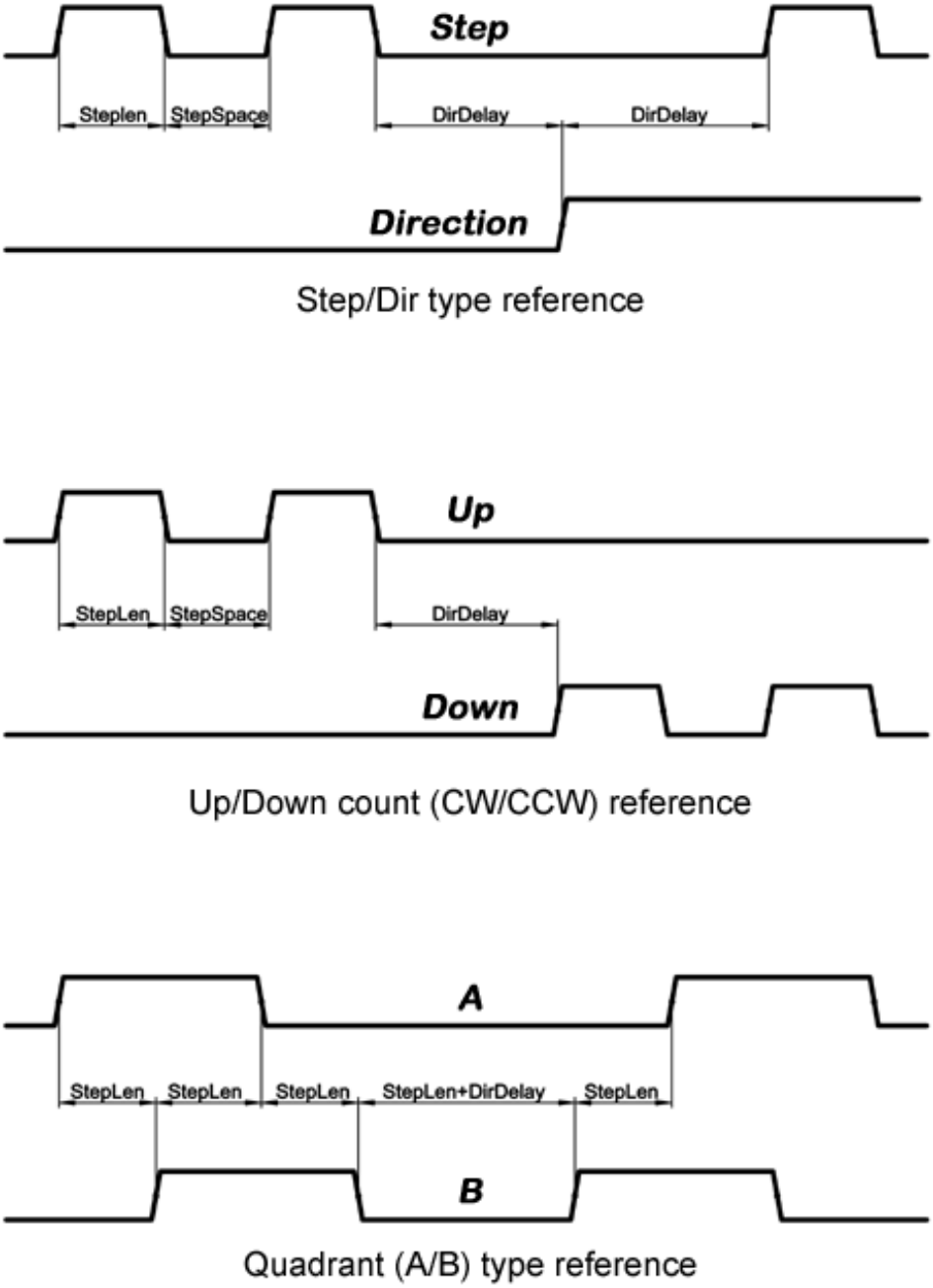


Figure 28.5: Reference signal timing diagrams

28.2.3.3 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 us
setp gm.0.stepgen.0.stepspace1000       # 1000 ns = 1 us
setp gm.0.stepgen.0.dirdelay 2000       # 2000 ns = 2 us
```

Connect stepgen to axis 0 position reference and enable pins:

```
net Xpos-cmd axis.0.motor-pos-cmd => gm.0.stepgen.0.position-cmd
net Xen axis.0.amp-enable-out => gm.0.stepgen.0.enable
```

28.2.4 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
```

28.2.4.1 Pins

Table 28.9: 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.

28.2.5 Axis DAC

The GM6-PCI motion control card has six serial axis DAC driver modules, one for each axis. 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 form 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
```

28.2.5.1 Pins

Table 28.10: 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.

28.2.5.2 Parameters

Table 28.11: 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.

28.3 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. Tha baud rate is 1 Mbit. The position commad 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 modifivation 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 form 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
```

28.3.1 Pins

Table 28.12: 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.

28.3.2 Parameters

Table 28.13: CAN module parameters

Parameters	Type and direction	Parameter description
.position-scale	(float, R/W)	Scale in per length unit.

28.4 Watchdog timer

Watchdog timer resets at function:

```
gm.<nr of card>.read
```

28.4.1 Pins

Table 28.14: 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.

28.4.2 Parameters

Table 28.15: 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.

28.5 End-, homing- and E-stop switches

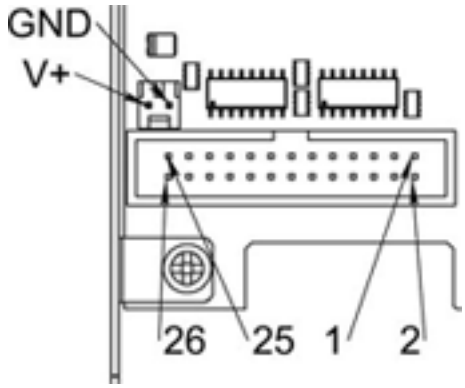


Figure 28.6: Pin numbering of homing & end switch connector

Table 28.16: 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 axis. All the names of these pins begin as follows:

```
gm.<nr. of card>.axis.<nr of axis>
```

,where nr of axis is form 0 to 5. For example:

```
gm.0.axis.0.home-sw-in
```

indicates the state of the axis 0 home switch.

Hal pins are updated by function:

```
gm.<nr of card>.read
```

28.5.1 Pins

Table 28.17: 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

28.5.2 Parameters

Table 28.18: 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

28.6 Status LEDs

28.6.1 CAN

Color: Orange

- Blink, during data communication.
- On, when any of the buffers are full - communication error.
- Off, when no data communication.

28.6.2 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.

28.6.3 EMC

Color: White

- Blink, when LinuxCNC is running.
- Otherwise off.

28.6.4 Boot

Color: Green

- On, when system booted successfully.
- Otherwise off.

28.6.5 Error

Color: Red

- Off, when there is no fault in the system.
- Blink, when PCI communication error.
- On, when watchdog timer overflowed.

28.7 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 sate (green LED blinking), and turns all outputs to error sate.

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.

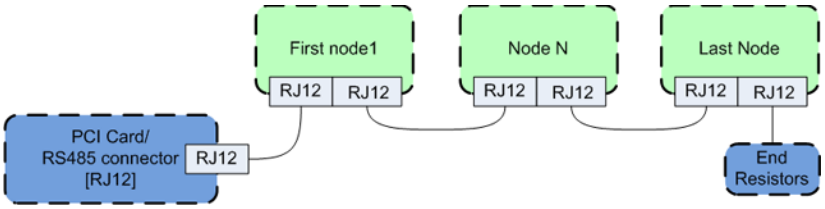


Figure 28.7: Connecting the RS485 nodes to the GM6-PCI card

Adressing:

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 jet identified, or when module is dropped down.
- Off, during identification (computer is on, but LinuxCNC not started)
- On, when it communicates continuously.

28.7.1 Relay output module

For pinout, connection and electrical charasteristics 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.<modul ID>
```

,where <modul ID> is form 00 to 15.

28.7.1.1 Pins

Table 28.19: Relay output module pins

Pins	Type and direction	Pin description
.relay-<0-7>	(bit, Out)	Output pin for relay

28.7.1.2 Parameters

Table 28.20: Relay output module parameters

Parameters	Type and direction	Parameter description
.invert-relay-<0-7>	(bit, R/W)	Negate relay output pin

28.7.1.3 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
```

28.7.2 Digital input module

For pinout, connection and electrical charasteristics 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.<modul ID>
```

,where <modul ID> is form 00 to 15.

28.7.2.1 Pins

Table 28.21: 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

28.7.2.2 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
```

28.7.3 DAC & ADC module

For pinout, connection and electrical charasteristics 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.<modul ID>
```

,where <modul ID> is form 00 to 15.

28.7.3.1 Pins

Table 28.22: 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.

28.7.3.2 Parameters

Table 28.23: 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.

28.7.3.3 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
```

28.7.4 Teach Pendant module

For pinout, connection and electrical charasteristics 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.<modul ID>
```

,where <modul ID> is form 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.

28.7.4.1 Pins

Table 28.24: 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

28.7.4.2 Parameters

Table 28.25: 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.

28.7.4.3 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
```

28.8 Errata

28.8.1 GM6-PCI card Errata

The revision number in this section refers to the revision of the GM6-PCI card device.

28.8.1.1 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.

Part V

Advanced Topics

Chapter 29

Python Interface

This is work in progress by Michael Haberler. Comments, fixes, and addenda are welcome, especially for PositionLogger (A bit of intent, purpose and usage would help here!)

29.1 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
- support for position logging (???)

29.2 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 FIXME: explore the exception code

`linuxcnc` also defines the `error` Python exception type to support error reporting.

29.3 Reading LinuxCNC status

Here is a Python fragment to explore the contents of the `linuxcnc.stat` object which contains some 8ß0+ values (run while linuxcnc is running for typical values):

```
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.

29.3.1 linuxcnc.stat attributes

acceleration
(returns float) - default acceleration, reflects the ini entry [TRAJ] DEFAULT_ACCELERATION.

active_queue
(returns int) - 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 True/False) - status of adaptive feedrate override (0/1).

ain
(returns tuple of floats) - current value of the analog input pins.

angular_units
(returns string) - reflects [TRAJ] ANGULAR_UNITS ini value.

aout
(returns tuple of floats) - current value of the analog output pins.

axes
(returns string) - reflects [TRAJ] AXES 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 integer) - block delete currently on/off.

command
(returns string) - currently executing command.

current_line
(returns integer) - currently executing line, int.

current_vel
(returns float) - current velocity in Cartesian space.

cycle_time
(returns string) - reflects [TRAJ] CYCLE_TIME ini value (FIXME is this right?).

debug
(returns integer) - debug flag.

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, in Cartesian space.

dout
(returns tuple of integers) - current value of the digital output pins.

dtg
(returns tuple of 9 floats) - remaining distance of current move, 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 integer) - trajectory planner enabled flag.

estop
(returns integer) - estop flag.

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_PAUSE,EXEC_WAITING_FOR_MOTION_AND_IO, EXEC_WAITING_FOR_DELAY, EXEC_WAITING_FOR_SYSTEM_CMD.

feed_hold_enabled
(returns integer) - enable flag for feed hold.

feed_override_enabled
(returns integer) - enable flag for feed override.

feedrate
(returns float) - current feedrate override.

file
(returns string) - currently executing gcode file.

flood
(returns integer) - flood enabled.

g5x_index
(returns string) - currently active coordinate system, G54=0, G55=1 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 16 integers) - currently active G-codes.

homed
(returns integer) - flag. 1 if homed.

id
(returns integer) - currently executing motion id.

inpos
(returns integer) - machine-in-position flag.

input_timeout
(returns integer) - 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_FINIS INTERP_ENDFILE, INTERP_FILE_NOT_OPEN, INTERP_ERROR. see src/emc/nml_intf/interp_return.hh

joint_actual_position
(returns tuple of floats) - actual joint positions.

joint_position
(returns tuple of floats) - Desired joint positions.

kinematics_type
(returns integer) - identity=1, serial=2, parallel=3, custom=4 .

limit
(returns tuple of integers) - axis limit masks. minHardLimit=1, maxHardLimit=2, minSoftLimit=4, maxSoftLimit=8.

linear_units
(returns string) - 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 [TRAJ] MAX_VELOCITY.

mcodes
(returns tuple of 10 integers) - currently active M-codes.

mist
(returns integer) - *mist on* flag.

motion_line
(returns integer) - source line number motion is currently executing. Relation to `id` unclear.

motion_mode
(returns integer) - motion mode.

motion_type
(returns integer) - trajectory planner mode. One of TRAJ_MODE_COORD, TRAJ_MODE_FREE, TRAJ_MODE_TELEOP.

optional_stop
(returns integer) - option stop flag.

paused
(returns integer) - `motion paused` flag.

pocket_prepped
(returns integer) - A Tx command completed, and this pocket is prepared. -1 if no prepared pocket.

poll()
- method to update current status attributes.

position
(returns tuple of floats) - trajectory position.

probe_tripped
(returns integer) - 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 integer) - flag, 1 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 integer) - the trajectory planner queue is full.

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 3 floats) - current interpreter settings. settings[0] = sequence number, settings[1] = feed rate, settings[2] = speed.

spindle_brake
(returns integer) - value of the spindle brake flag.

spindle_direction
(returns integer) - rotational direction of the spindle. forward=1, reverse=-1.

spindle_enabled
(returns integer) - value of the spindle enabled flag.

spindle_increasing
(returns integer) - unclear.

spindle_override_enabled
(returns integer) - value of the spindle override enabled flag.

spindle_speed
(returns float) - spindle speed value, rpm, > 0: clockwise, < 0: counterclockwise.

spindlerate
(returns float) - spindle speed override scale.

rapidrate
(returns float) - rapid override scale.

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_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.

velocity
(returns float) - default velocity. reflects [TRAJ] DEFAULT_VELOCITY.

29.3.2 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:

```
import linuxcnc
s = linuxcnc.stat()
s.poll()
print 'Axis 1 homed: ', s.axis[1]['homed']
```

For each axis, the following dictionary keys are available:

axisType
(returns integer) - type of axis configuration parameter, reflects [AXIS_x]TYPE. LINEAR=1, ANGULAR=2. See Axis ini configuration for details.

backlash
(returns float) - Backlash in machine units. configuration parameter, reflects [AXIS_x]BACKLASH.

enabled
(returns integer) - non-zero means enabled.

fault
(returns integer) - non-zero means axis amp fault.

ferror_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.

max_ferror
(returns float) - maximum following error. configuration parameter, reflects [AXIS_x]FERROR.

max_hard_limit
(returns integer) - non-zero means max hard limit exceeded.

max_position_limit
(returns float) - maximum limit (soft limit) for axis motion, in machine units.configuration parameter, reflects [AXIS_x]MAX_LIM

max_soft_limit
non-zero means `max_position_limit` was exceeded, int

min_ferror
(returns float) - configuration parameter, reflects [AXIS_x]MIN_FERROR.

min_hard_limit
(returns integer) - non-zero means min hard limit exceeded.

min_position_limit
(returns float) - minimum limit (soft limit) for axis motion, in machine units.configuration parameter, reflects [AXIS_x]MIN_LIMI

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) - units per mm, deg for linear, angular

velocity
(returns float) - current velocity.

29.4 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:

```
import linuxcnc
s = linuxcnc.stat()
c = linuxcnc.command()

def ok_for_mdi():
    s.poll()
    return not s.estop and s.enabled and s.homed 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")
```

29.5 Sending commands through `linuxcnc.command`

Before sending a command, initialize a command channel like so:

```
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, axis)
c.jog(linuxcnc.JOG_CONTINUOUS, axis, speed)
c.jog(linuxcnc.JOG_INCREMENT, axis, speed, 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)
```

29.5.1 `linuxcnc.command` attributes

`serial`
the current command serial number

29.5.2 `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.

`feedrate(float)`
set the feedrate.

`flood(int)`
turn on/off flooding.

`home(int)`
home a given axis.

`jog(int, int, [, int[,int]])`
Syntax:
jog(command, axis[, velocity[, distance]])
jog(linuxcnc.JOG_STOP, axis)
jog(linuxcnc.JOG_CONTINUOUS, axis, velocity)
jog(linuxcnc.JOG_INCREMENT, axis, velocity, distance)
Constants:
JOG_STOP (0)
JOG_CONTINUOUS (1)
JOG_INCREMENT (2)

`load_tool_table()`
reload the tool table.

`maxvel(float)`
set maximum velocity

`mdi(string)`
send an MDI command. Maximum 255 chars.

mist(int)
turn on/off mist.
Syntax:
mist(command)
mist(linuxcnc.MIST_ON) [(1)]
mist(linuxcnc.MIST_OFF) [(0)]
Constants:
MIST_ON (1)
MIST_OFF (0)

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.

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)
set spindle override flag

spindle(int)
set spindle direction. Argument one of SPINDLE_FORWARD, SPINDLE_REVERSE, SPINDLE_OFF, SPINDLE_INCREASE, SPINDLE_DECREASE, or SPINDLE_CONSTANT.

spindleoverride(float)
set spindle override factor

state(int)
set the machine state. Machine state should be STATE_ESTOP, STATE_ESTOP_RESET, STATE_ON, or STATE_OFF

teleop_enable(int)
enable/disable teleop mode.

teleop_vector(float, float, float [,float, float, float])
set teleop destination vector

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 axis.

wait_complete([float])
wait for completion of the last command sent. If timeout in seconds not specified, default is 1 second.

29.6 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.

```
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
```

29.7 Reading ini file values

Here's an example for reading values from an ini file through the `linuxcnc.ini` object:

```
# run as:
# python ini-example.py ~/emc2-dev/configs/sim/axis/axis_mm.ini

import sys
import linuxcnc

inifile = linuxcnc.ini(sys.argv[1])

# inifile.find() returns None if the key wasnt found - the
# following idiom is useful for setting a default value:

machine_name = inifile.find('EMC', 'MACHINE') or "unknown"
print "machine name: ", machine_name

# inifile.findall() returns a list of matches, or an empty list
# if the key wasnt found:

extensions = inifile.findall("FILTER", "PROGRAM_EXTENSION")
```

```
print "extensions: ", extensions

# override default NML file by ini parameter if given
nmlfile = inifile.find("EMC", "NML_FILE")
if nmlfile:
    linuxcnc.nmlfile = os.path.join(os.path.dirname(sys.argv[1]), nmlfile)
```

29.8 The `linuxcnc.positionlogger` type

Some usage hints can be gleaned from `src/emc/usr_intf/gremlin/gremlin.py`.

29.8.1 members

`npts`
number of points.

29.8.2 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 ,

Chapter 30

Kinematics

30.1 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]

30.1.1 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).

30.2 Trivial Kinematics

The simplest machines are those in which 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 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];
pos->a = joints[3];
pos->b = joints[4];
pos->c = joints[5];
```

[1] The word “axes” is also commonly (and wrongly) used when talking about CNC machines, and referring to the moving directions of the machine.
[2] Kinematics: a two way function to transform from Cartesian space to joint space

In the above code snippet one can see how the mapping is done: the X position is identical with the joint 0, the Y posittion with 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;
joints[3] = pos->a;
joints[4] = pos->b;
joints[5] = pos->c;
```

As one can see, it's pretty straightforward to do the transformation for a trivial "kins" (kinematics) or Cartesian machine. It gets a bit more complicated if the machine is missing one of the axes.[3] [4]

30.3 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).

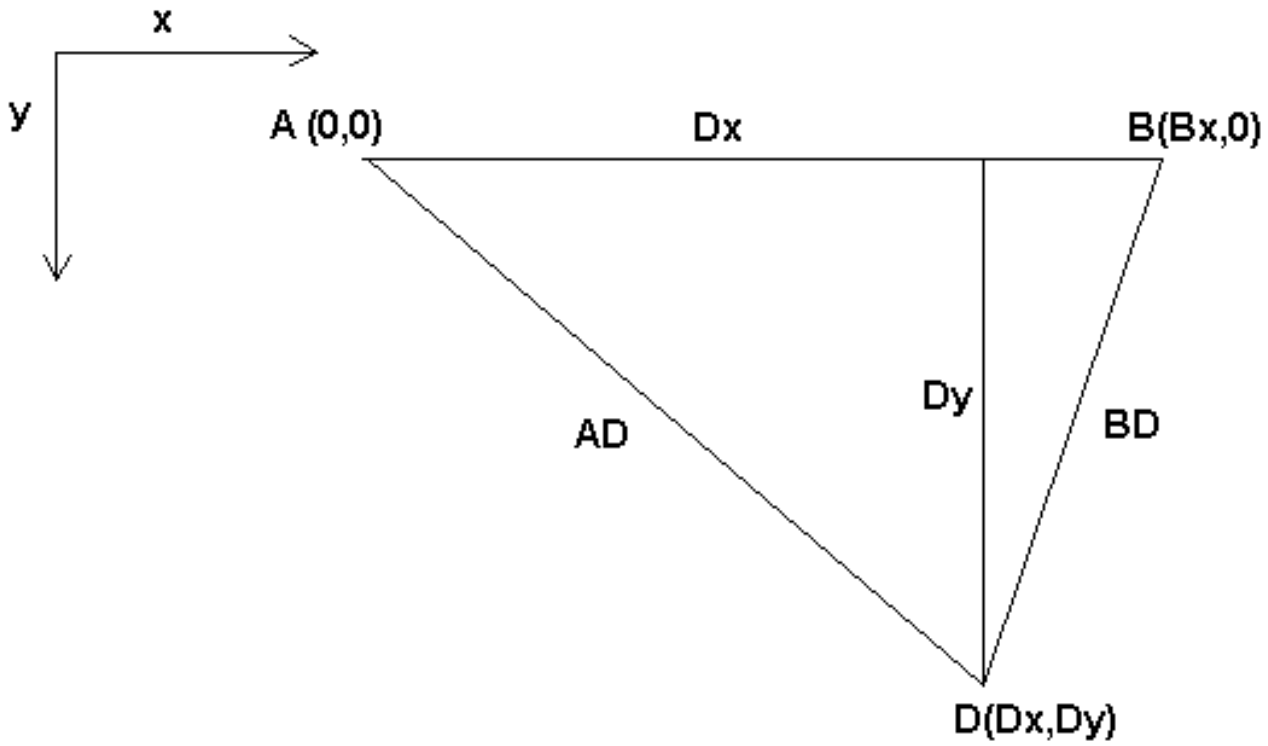


Figure 30.1: Bipod setup

[3] If a machine (e.g. a lathe) is set up with only the axes X,Z & A, and the LinuxCNC inifile holds only these 3 joints defined, then the above matching will be faulty. That is because we actually have (joint0=x, joint1=Z, joint2=A) whereas the above assumes joint1=Y. To make it easily work in LinuxCNC one needs to define all axes (XYZA), then use a simple loopback in HAL for the unused Y axis.

[4] One other way of making it work, is by changing the matching code and recompiling the software.

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.

30.3.1 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 $AD^2 = x^2 + y^2$ $BD^2 = (Bx - x)^2 + y^2$, 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:

$$y = \sqrt{AD2 - x2}$$

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;
```

30.3.2 Inverse transformation

The inverse kinematics is lots 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;
```

30.4 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, typically *KINEMATICS_BOTH*.

```
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 *comp*. See the *comp(1)* manpage or the HAL manual for more information.

Chapter 31

Stepper Tuning

31.1 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.

31.1.1 Run a Latency Test

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 *ovl max* 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 10636 nano-seconds, or 10.6 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 us latency. Fortunately that is fixable, see FixingDapperSMIIssues in the wiki found at wiki.linuxcnc.org.

So, what do the results mean? If your *ovl max* 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 EMC, 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 Matrox video card solved the problem - EMC does not require bleeding edge hardware.

31.1.2 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 us min (Step on falling edge)
Step Pulse "1" Time: 4.5 us min
Direction Setup: 1 us min (20 us min hold time after Step edge)
```

From the Gecko G203V manual:

```
Step Frequency: 0 to 333 kHz
Step Pulse "0" Time: 2.0 us min (Step on rising edge)
Step Pulse "1" Time: 1.0 us min

Direction Setup:
    200 ns (0.2 us) before step pulse rising edge
    200 ns (0.2 us) 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 us
Minimum STEP pulse low time 1.0 us
Step happens on rising edge
```

Once you find the numbers, write them down too - you need them in the next step.

31.1.3 Choose your BASE_PERIOD

BASE_PERIOD is the *heartbeat* of your EMC 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 us and high for 4.5 us, it needs the direction pin to be stable 1 us before the falling edge, and remain stable for 20 us after the falling edge. The longest timing requirement is the 20 us hold time. A simple approach would be to set the period at 20 us. That means that all changes on the STEP and DIR lines are separated by 20 us. All is good, right?

Wrong! If there was ZERO latency, then all edges would be separated by 20 us, and everything would be fine. But all computers have some latency. Latency means lateness. If the computer has 11 us of latency, that means sometimes the software runs as much as 11 us later than it was supposed to. If one run of the software is 11 us late, and the next one is on time, the delay from the first to the second is only 9 us. If the first one generated a step pulse, and the second one changed the direction bit, you just violated the 20 us 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 us hold time requirement, and your latency test said you have a maximum latency of 11 us, then if you set the BASE_PERIOD to 20+11 = 31 us (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 us, it takes 2x31 = 62 us 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 us). The longest time is the 2 us high time. If you have 11 us latency, then you can set the BASE_PERIOD as low as 11+2=13 us. Getting rid of the long 20 us hold time really helps! With a period of 13 us, a complete step takes 2x13 = 26 us, and the maximum step rate is 38,461 steps per second!

But you can't start celebrating yet. Note that 13 us is a very short period. If you try to run the step generator every 13 us, 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 us, you should start at 25 us or more, run EMC, 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, sppose you started at 25 us, trying to get to 13 us, but you find that around 16 us is the limit - any less and the computer doesn't respond very well. So you use 16 us. With a 16 us period and 11 us latency, the shortest output time will be 16-11 = 5 us. The drive only needs 2 us, 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 us for the period, so a step takes 32 us. That works out to a not bad 31,250 steps per second.

31.1.4 Use steplen, stepspace, dirsetup, and/or dirhold

In the last section, we got the Xylotex drive to a 16 us period and a 31,250 step per second maximum speed. But the Gecko was stuck at 31 us 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 us hold time requirement. That plus the 11 us latency is what forces us to use a slow 31 us 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 it 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 us hold time requirement, then the next longest time is the 4.5 us high time. Add the 11 us latency to the 4.5 us high time, and you get a minimum period of 15.5 us. When you try 15.5 us, you find that the computer is sluggish, so you settle on 16 us. If we leave dirhold at 1 (the default), then the minimum time between step and direction is the 16 us period minus the 11 us latency = 5 us, which is not enough. We need another 15 us. Since the period is 16 us, 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 us, 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 us, then a combination of a 16 us 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 us to make each step, and we have the same 31,250 step per second rate that we got with the Xylotex.

The 11 us latency number used in this example is very good. If you work through these examples with larger latency, like 20 or 25 us, 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.

31.1.5 No Guessing!

For a fast AND reliable software based stepper system, you cannot just guess at periods and other configuration paremeters. 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 http://wiki.linuxcnc.org/uploads/StepTimingCalculator.ods 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.

Chapter 32

PID Tuning

32.1 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).

32.1.1 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.

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).

[1] This Subsection is taken from an much more extensive article found at http://en.wikipedia.org/wiki/PID_controller

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.

32.1.2 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:

32.1.2.1 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.

32.1.2.2 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.

32.1.2.3 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.

32.1.3 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.

32.1.3.1 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

32.1.3.2 Ziegler-Nichols method

Another tuning method is formally known as the *Ziegler-Nichols method*, introduced by John G. Ziegler and Nathaniel B. Nichols. 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	$.6K_c$	$P_c/2$	$P_c/8$

32.1.3.3 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.

Part VI

Ladder Logic

Chapter 33

Classicladder Introduction

33.1 History

Classic Ladder 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 Classic Ladder website*

Classic Ladder 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 Enhanced Machine Controller project.

33.2 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

33.3 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:

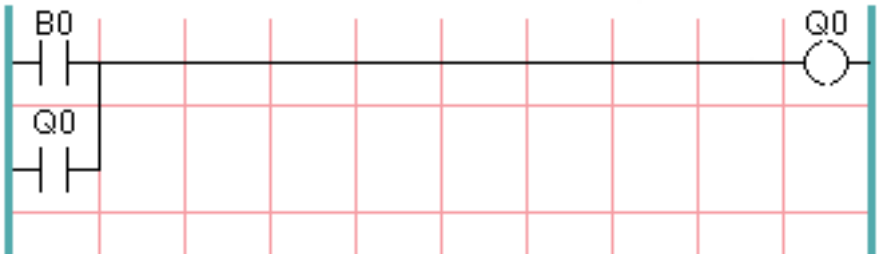
The input on the left, B0, a normally open contact, is connected to the coil (output) on the right, Q0. Now imagine a voltage gets applied to the leftmost end, because the input B0 turns true (e.g. the input is activated, or the user pushed the NO contact). The voltage has a direct path to reach the 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.

33.4 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 auxilliary contacts in addition to the large 3-phase contacts that are the primary feature of the contactor.

Since this auxilliary 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 auxilliary contact Q0 and connect it across the B0 *pushbutton* contact from our earlier example.

Let's take a look at it:

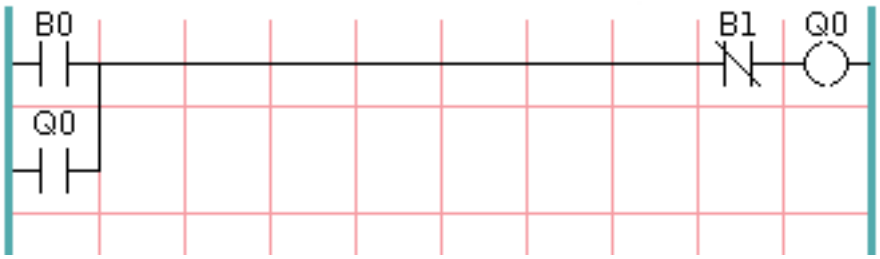


As before, when the user presses pushbutton B0, coil Q0 comes on. And when coil Q0 comes on, switch Q0 comes on. Now the interesting part happens. When the user releases pushbutton B0, coil Q0 does not stop as it did before. This is because switch Q0 of this circuit is effectively holding the user's 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.

Chapter 34

Classicladder Programming

34.1 Ladder Concepts

Classic Ladder 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.

34.2 Languages

The most common language used when working with Classic Ladder is *ladder*. Classic Ladder also supports Sequential Function Chart (Grafcet).

34.3 Components

There are 2 components to Classic Ladder.

- The real time module classicladder_rt
- The user space module (including a GUI) classicladder

34.3.1 Files

Typically classic ladder 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.

34.3.2 Realtime Module

Loading the Classic Ladder real time module (classicladder_rt) is possible from a HAL file, or directly using a halcmd instruction. The first line loads real time the Classic Ladder module. The second line adds the function classicladder.0.refresh to the servo thread. This line makes Classic Ladder update at the servo thread rate.

```
loadrt classicladder_rt
addf classicladder.0.refresh servo-thread
```

The speed of the thread that Classic Ladder is running in directly affects the responsiveness to inputs and outputs. If you can turn a switch on and off faster than Classic Ladder can notice it then you may need to speed up the thread. The fastest that Classic Ladder 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 Classic Ladder 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.

34.3.3 Variables

It is possible to configure the number of each type of ladder object while loading the Classic Ladder real time module. If you do not configure the number of ladder objects Classic Ladder will use the default values.

Table 34.1: 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
```

34.4 Loading the Classic Ladder user module

Classic Ladder HAL commands must executed before the GUI loads or the menu item Ladder Editor will not function. If you used the Stepper Config Wizard place any Classic Ladder 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 use Sections.

To load a ladder file:

```
loadusr classicladder myladder.clp
```

Classic Ladder 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 Classic Ladder with HAL without EMC:

```
loadusr –w classicladder
```

The -w tells HAL not to close down the HAL environment until Classic Ladder is finished.

If you first load ladder program with the *--nogui* option then load Classic Ladder again with no options the GUI will display the last loaded ladder program.

In AXIS you can load the GUI from File/Ladder Editor…

34.5 Classic Ladder GUI

If you load Classic Ladder with the GUI it will display two windows: section display, and section manager.

34.5.1 Sections Manager

When you first start up Classic Ladder you get an empty Sections Manager window.

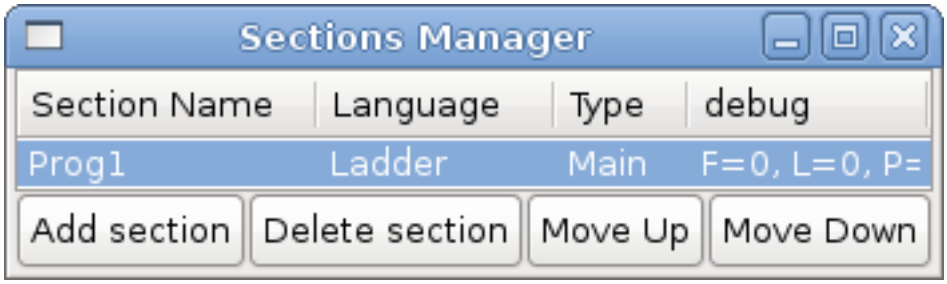


Figure 34.1: Sections Manager Default Window

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.

34.5.2 Section Display

When you first start up Classic Ladder you get an empty Section Display window. Displayed is one empty rung.

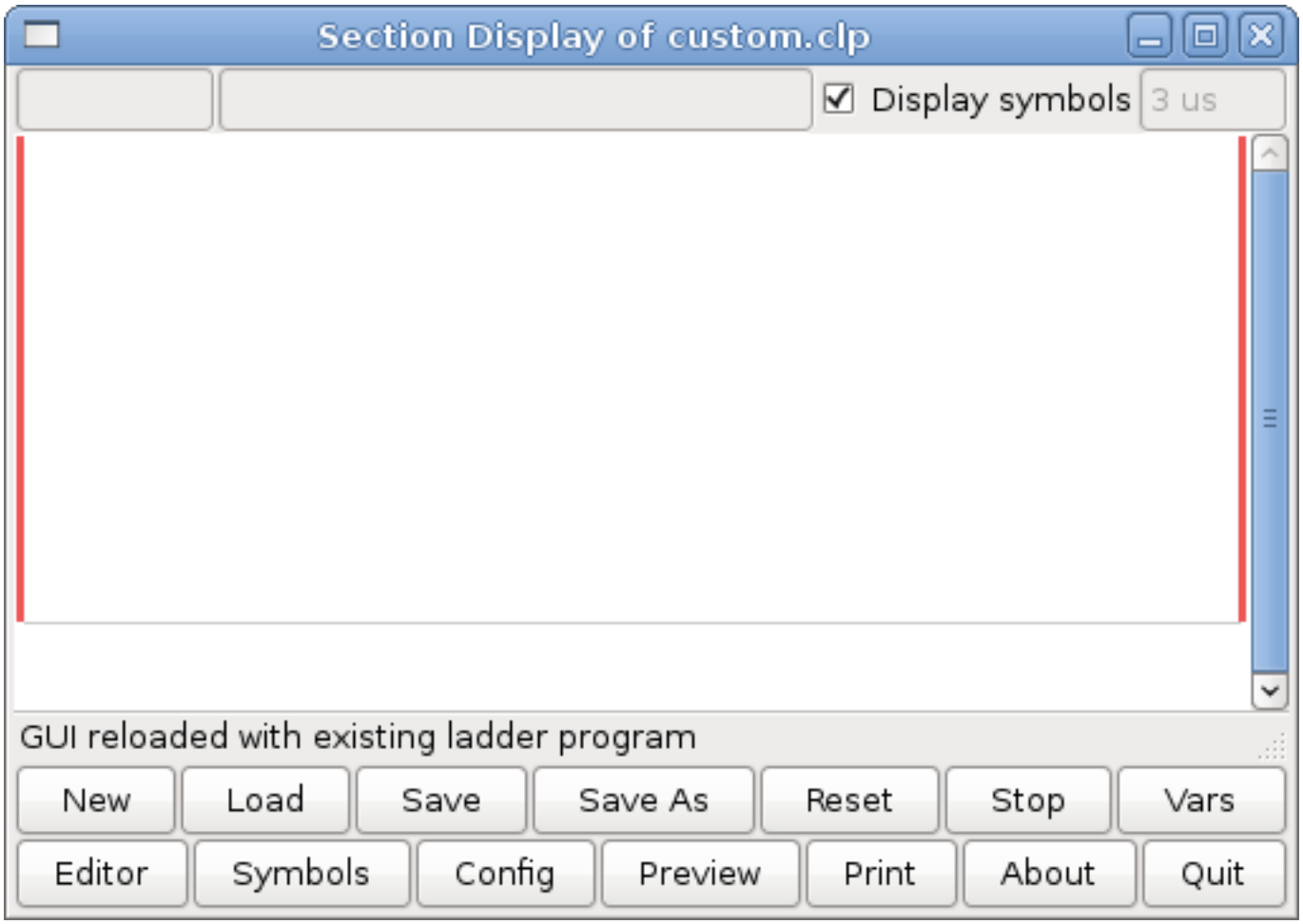


Figure 34.2: Section Display Default Window

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.

34.5.3 The Variable Windows

This 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.

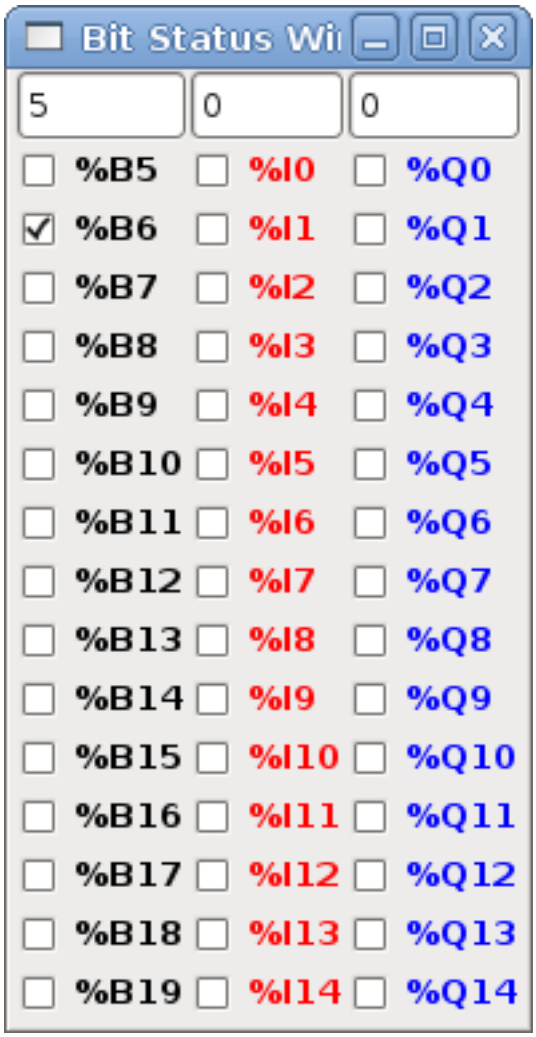


Figure 34.3: Bit Status Window

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 Classic Ladder is running can not be changed and will be displayed as checked if on and unchecked if off.

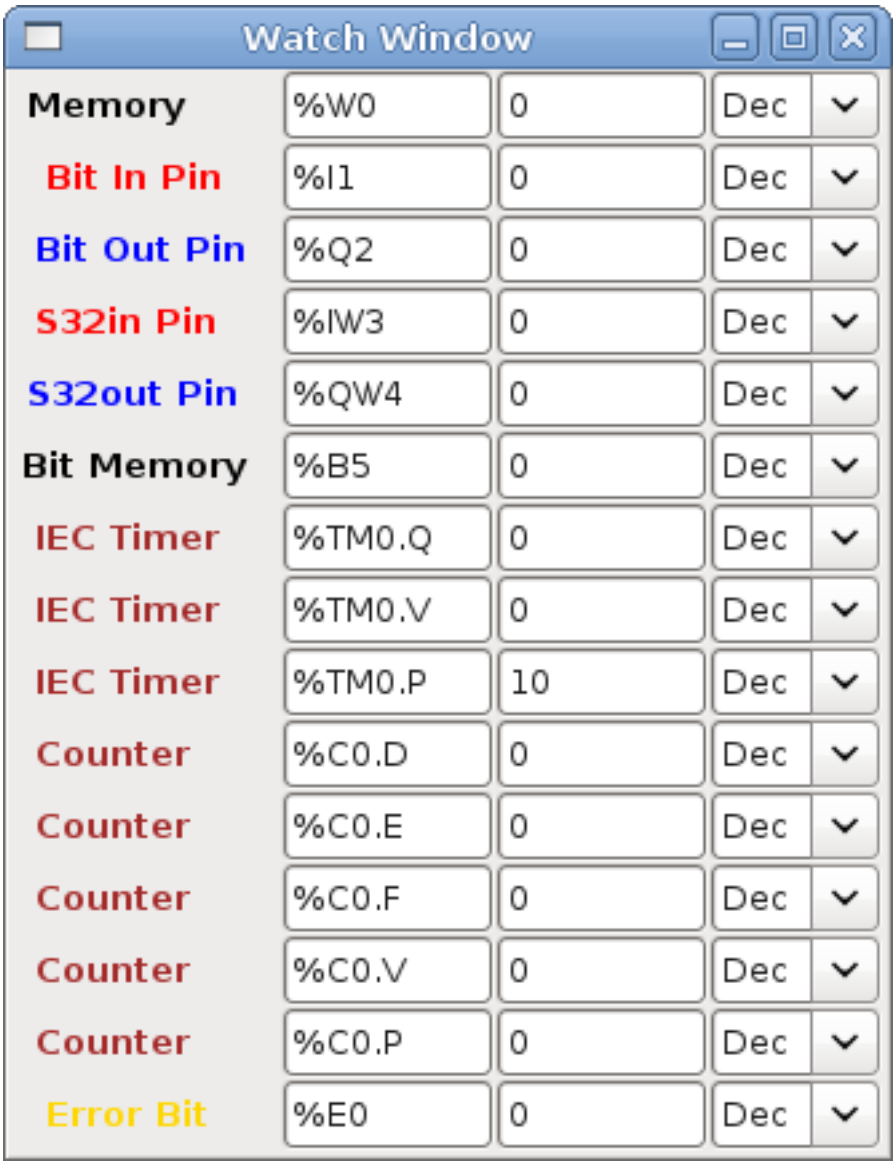


Figure 34.4: Watch Window

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.

34.5.4 Symbol Window

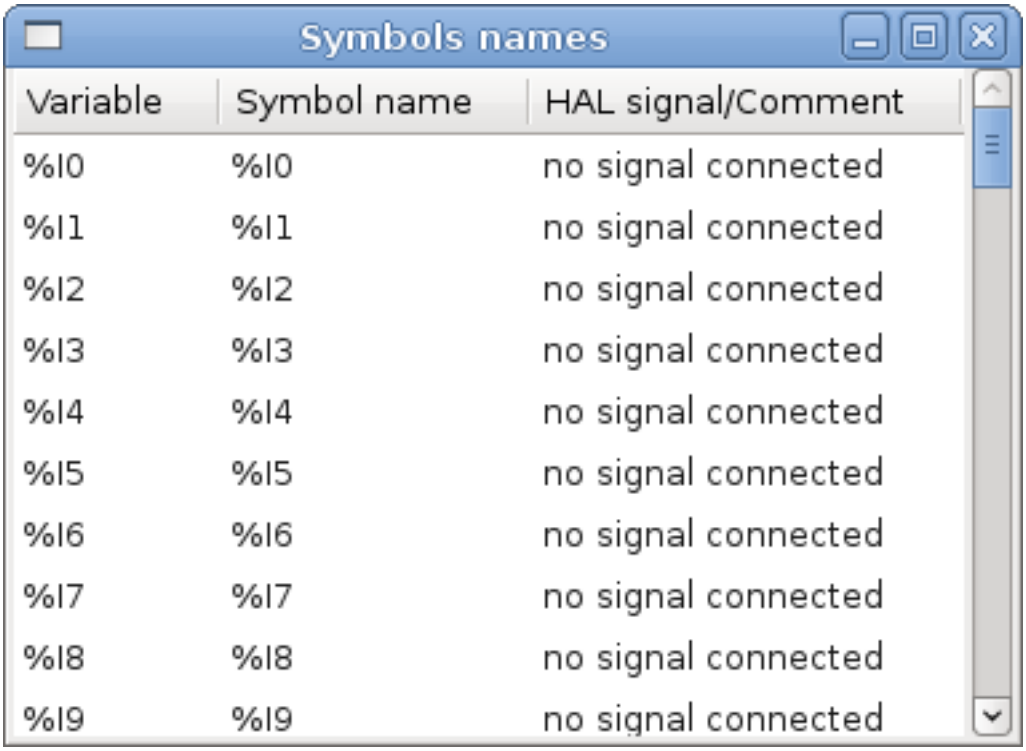


Figure 34.5: Symbol Names window

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!

34.5.5 The Editor window

Figure 34.6: 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.

34.5.6 Config Window

The config window shows the current project status and has the Modbus setup tabs.

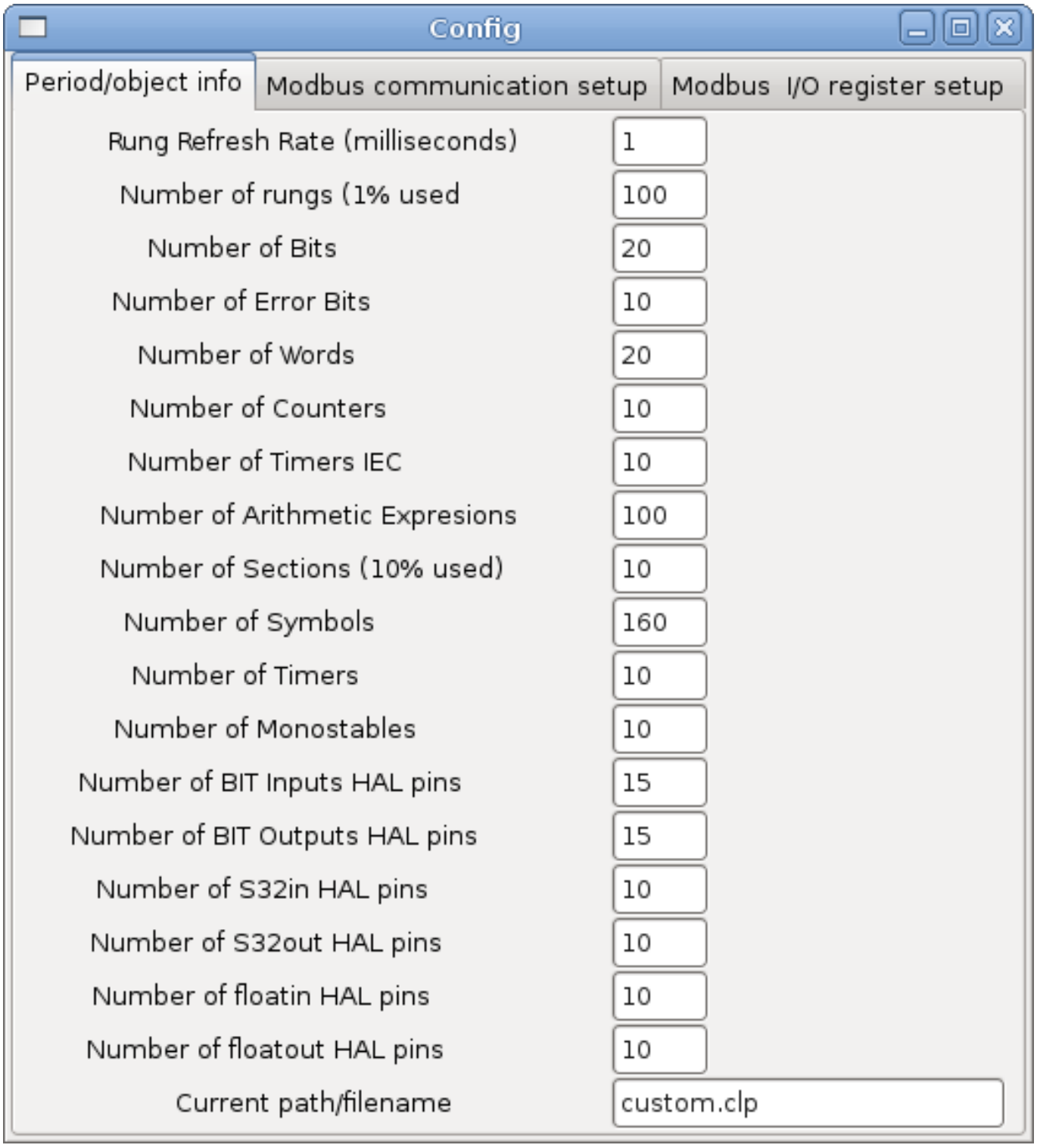


Figure 34.7: Config Window

34.6 Ladder objects

34.6.1 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.

34.6.2 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)

34.6.3 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)

34.6.4 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)

34.6.5 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)

34.6.6 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)
- (,) grouping
- ^ (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 > %CO.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.

34.6.7 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 (0x07FFFFFF) (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.

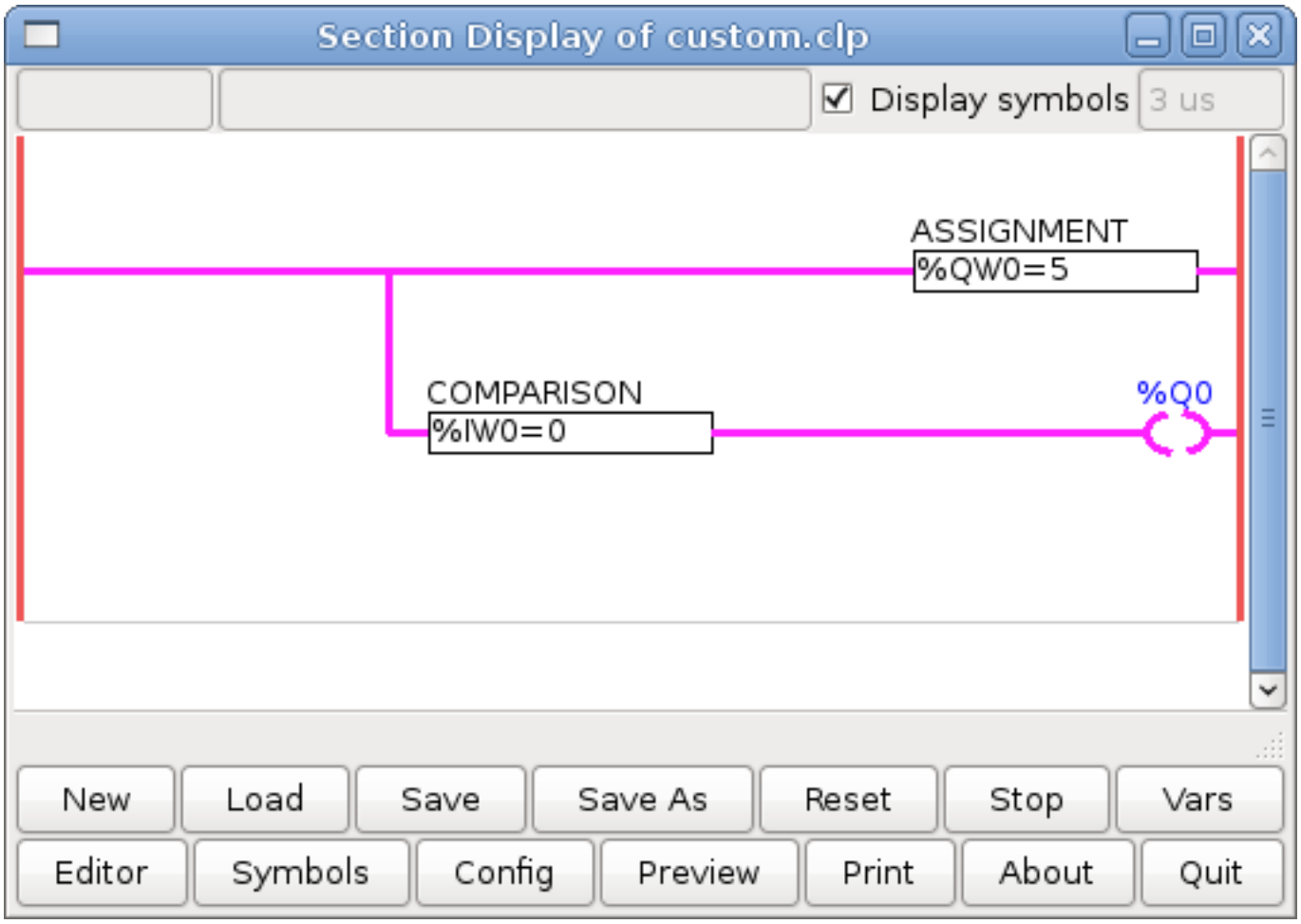


Figure 34.8: Assign/Compare Example

34.6.8 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.0. 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!

34.6.8.1 JUMP COIL

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.

34.6.8.2 CALL COIL

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.

34.7 Classic Ladder 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 Classic Ladder**)
- *%QFxx* - Word memory xx (Float out pin) (**converted to S32 in Classic Ladder**)
- *%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

34.8 GRAFCET Programming

Warning
This is probably the least used and most poorly understood feature of Classic Ladder. 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 image: 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 [show sequential program]

- *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.

34.9 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 Classic Ladder 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 Classic Ladder userspace program.

Loading Modbus

```
loadusr -w classicladder --modmaster myprogram.clp
```

The -w makes HAL wait until you close Classic Ladder before closing realtime session. Classic Ladder 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 Classic Ladder user program this page will not be displayed.

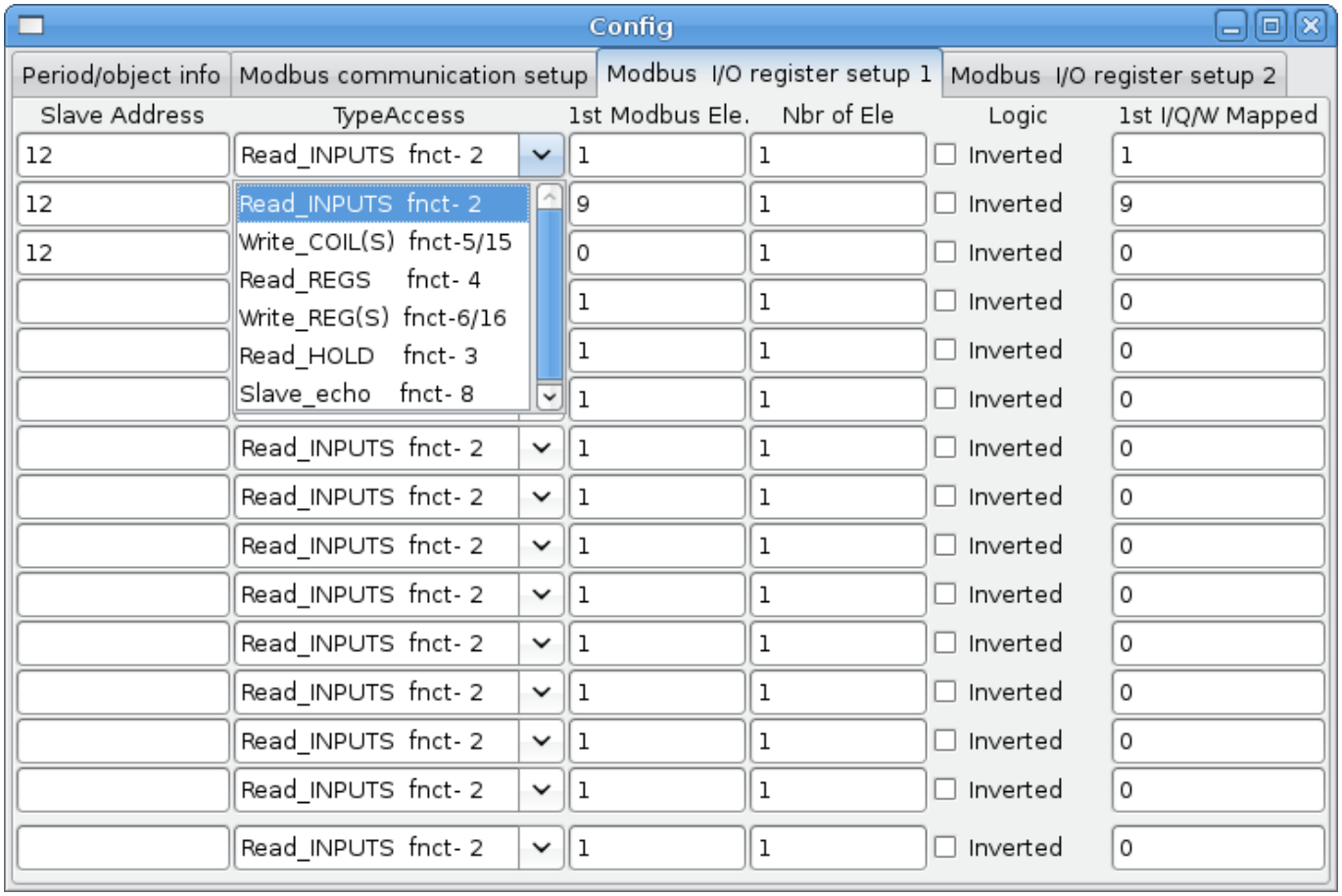


Figure 34.9: Config I/O

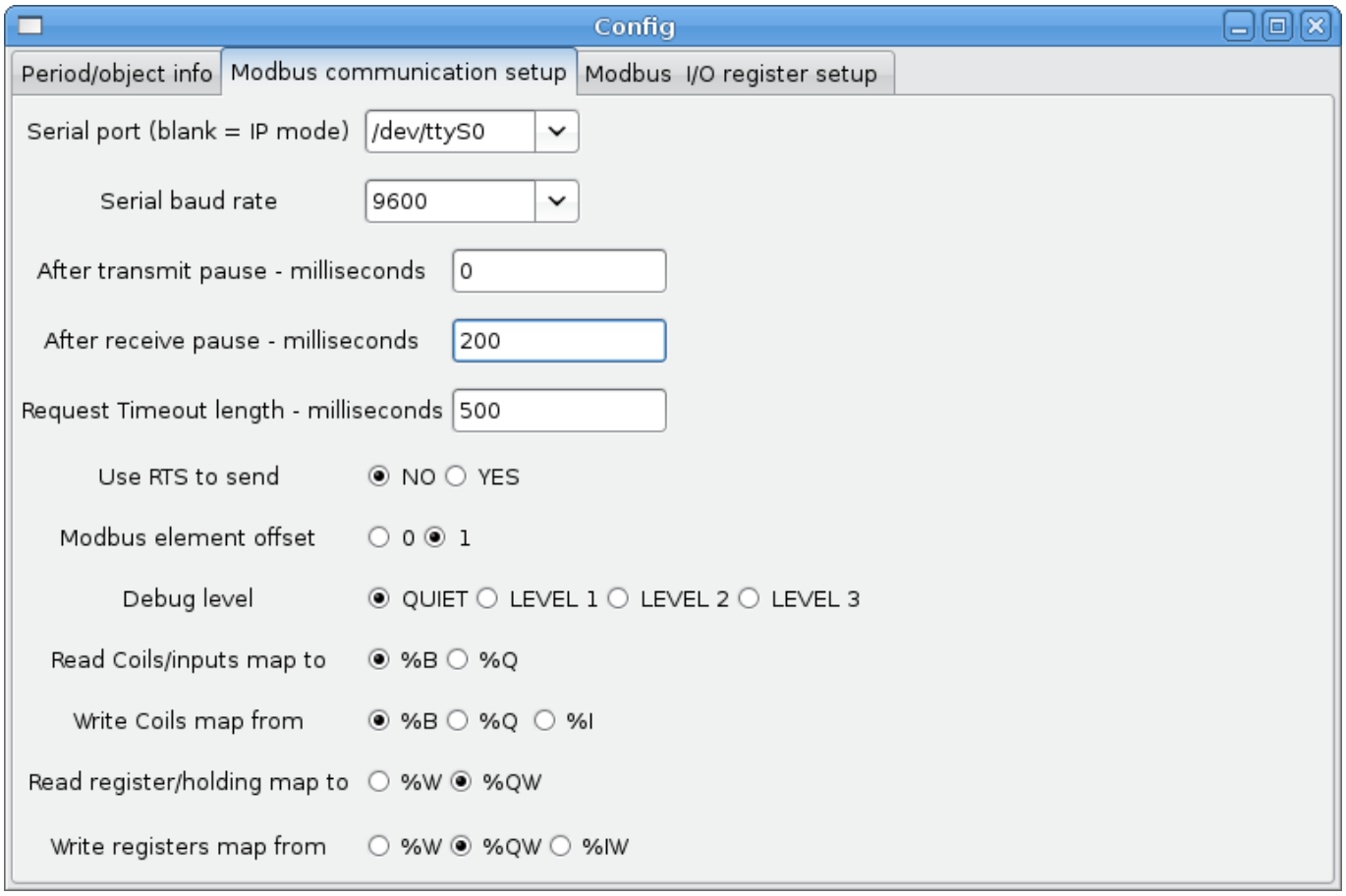


Figure 34.10: Config Coms

- *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 Classic Ladder'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 Classic Ladder'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 Classic Ladder'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 then 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 Classic Ladder running in a terminal to see the message.

34.9.1 MODBUS Settings

Serial:

- Classic Ladder 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. Classic Ladder 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).

34.9.2 MODBUS Info

- Classic Ladder 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
- Classic Ladder 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/

34.9.3 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.

34.9.4 MODBUS Bugs

- In compare blocks the function %W=ABS(%W1-%W2) is accepted but does not compute properly. only %W0=ABS(%W1) is currently legal.
- When loading a ladder program it will load Modbus info but will not tell Classic Ladder 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.

34.10 Setting up Classic Ladder

In this section we will cover the steps needed to add Classic Ladder to a Stepconf Wizard generated config. On the advanced Configuration Options page of Stepconf Wizard check off "Include Classic Ladder PLC".

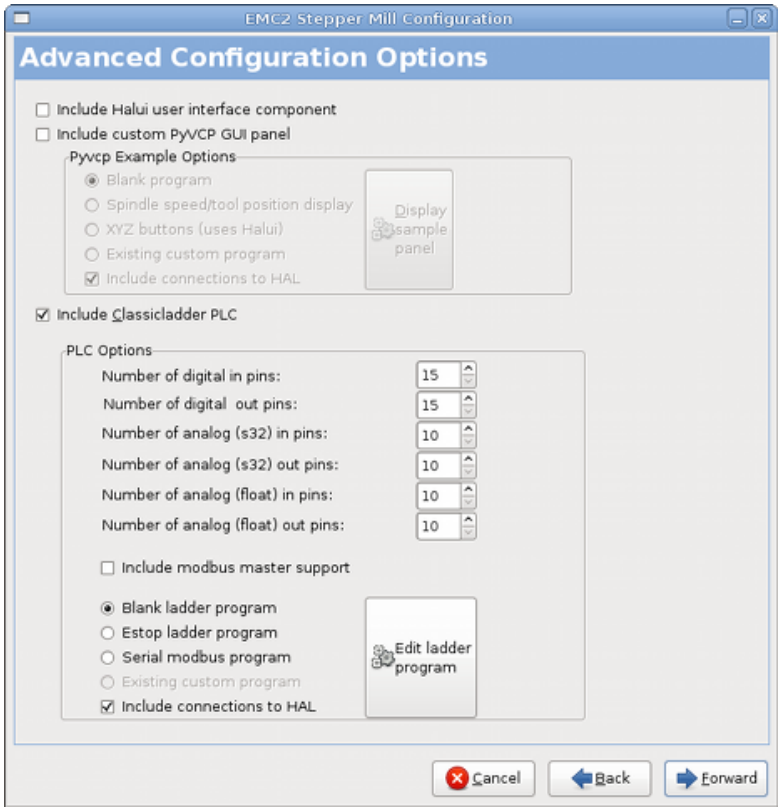


Figure 34.11: Stepconf Classic Ladder

34.10.1 Add the Modules

If you used the Stepconf Wizard to add Classic Ladder you can skip this step.

To manually add Classic Ladder 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 Classic Ladder function to the servo thread:

```
addf classicladder.0.refresh servo-thread
```

34.10.2 Adding Ladder Logic

Now start up your config and select "File/Ladder Editor" to open up the Classic Ladder 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.

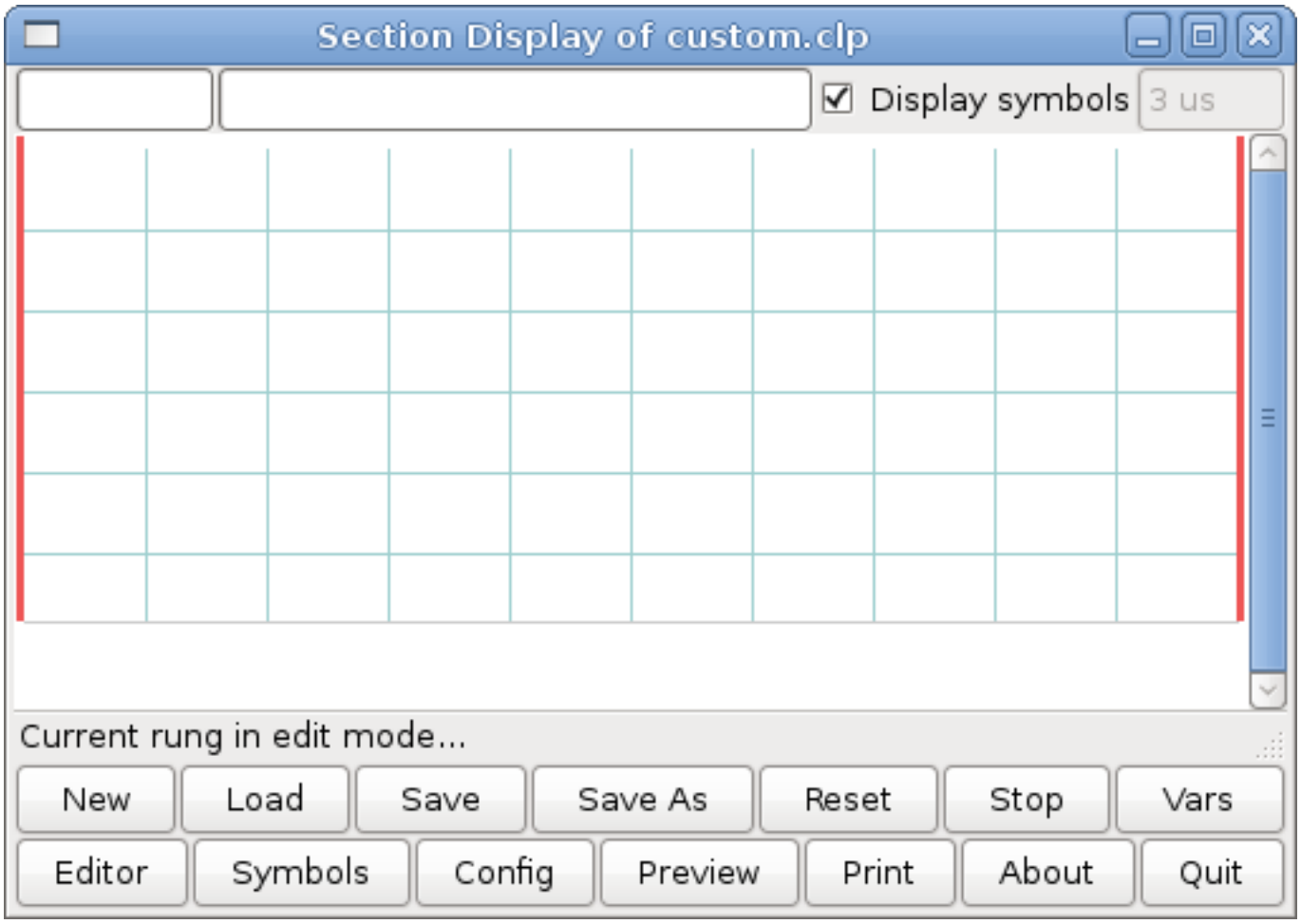


Figure 34.12: Section Display with Grid

Now click on the N.O. Input in the Editor Window.

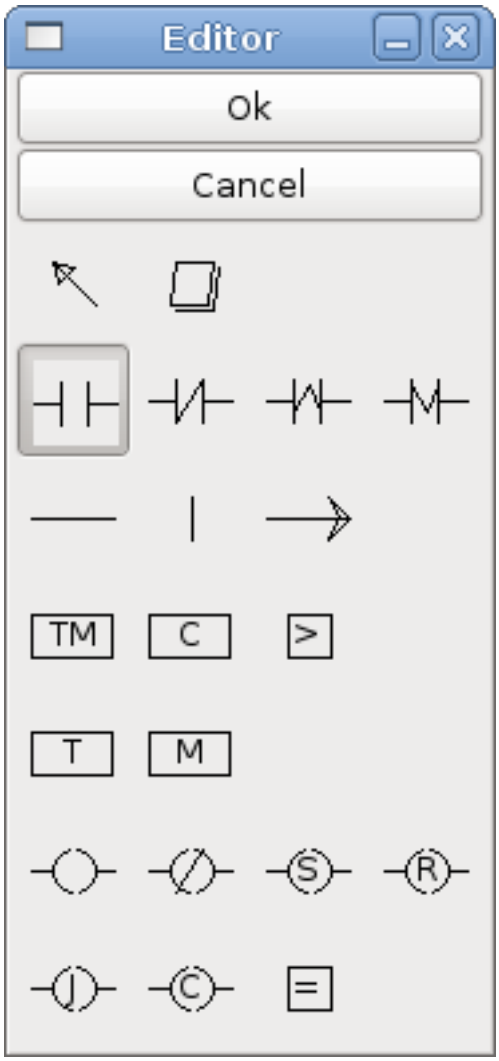


Figure 34.13: Editor Window

Now click in the upper left grid to place the N.O. Input into the ladder.

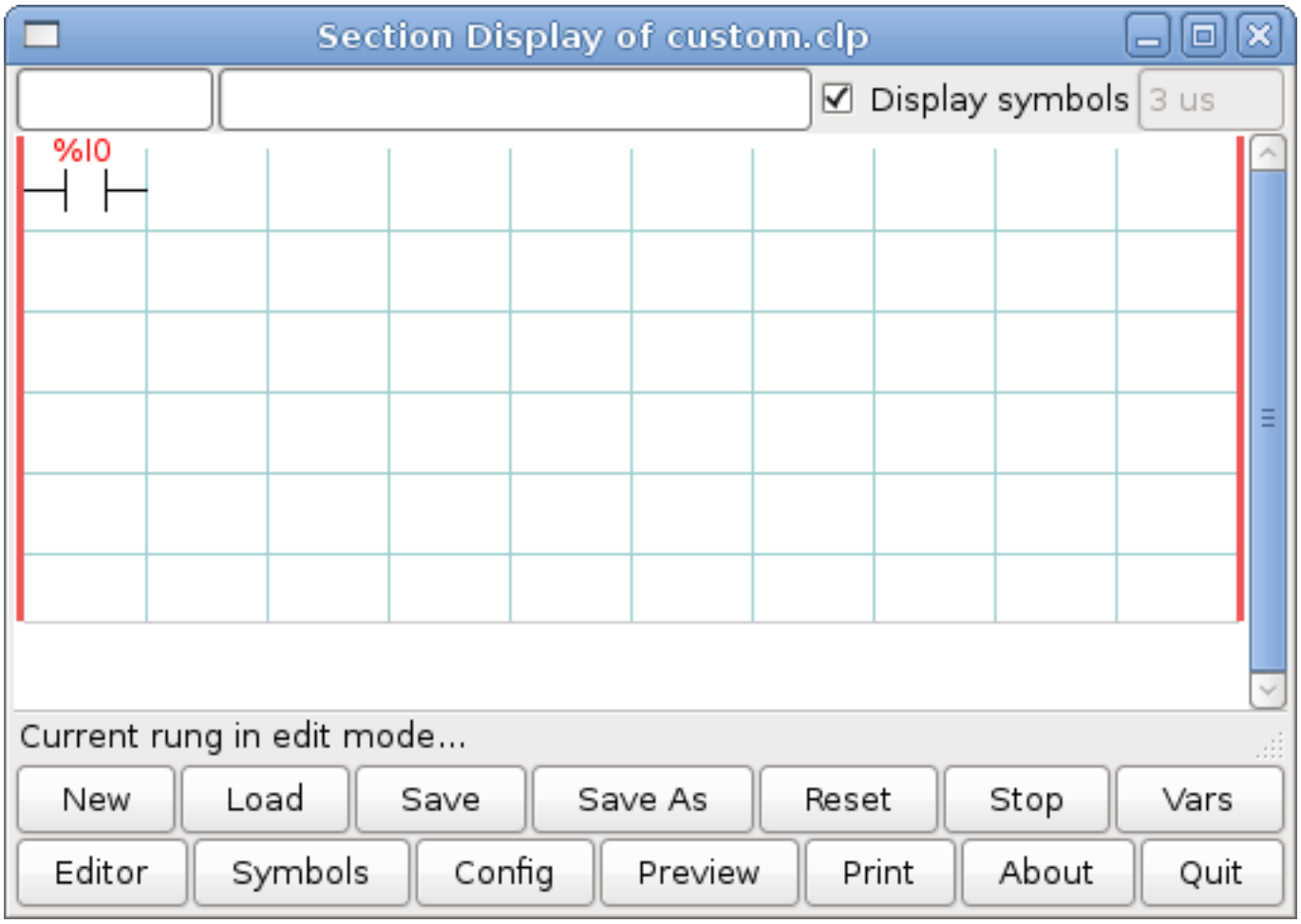


Figure 34.14: Section Display with Input

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.

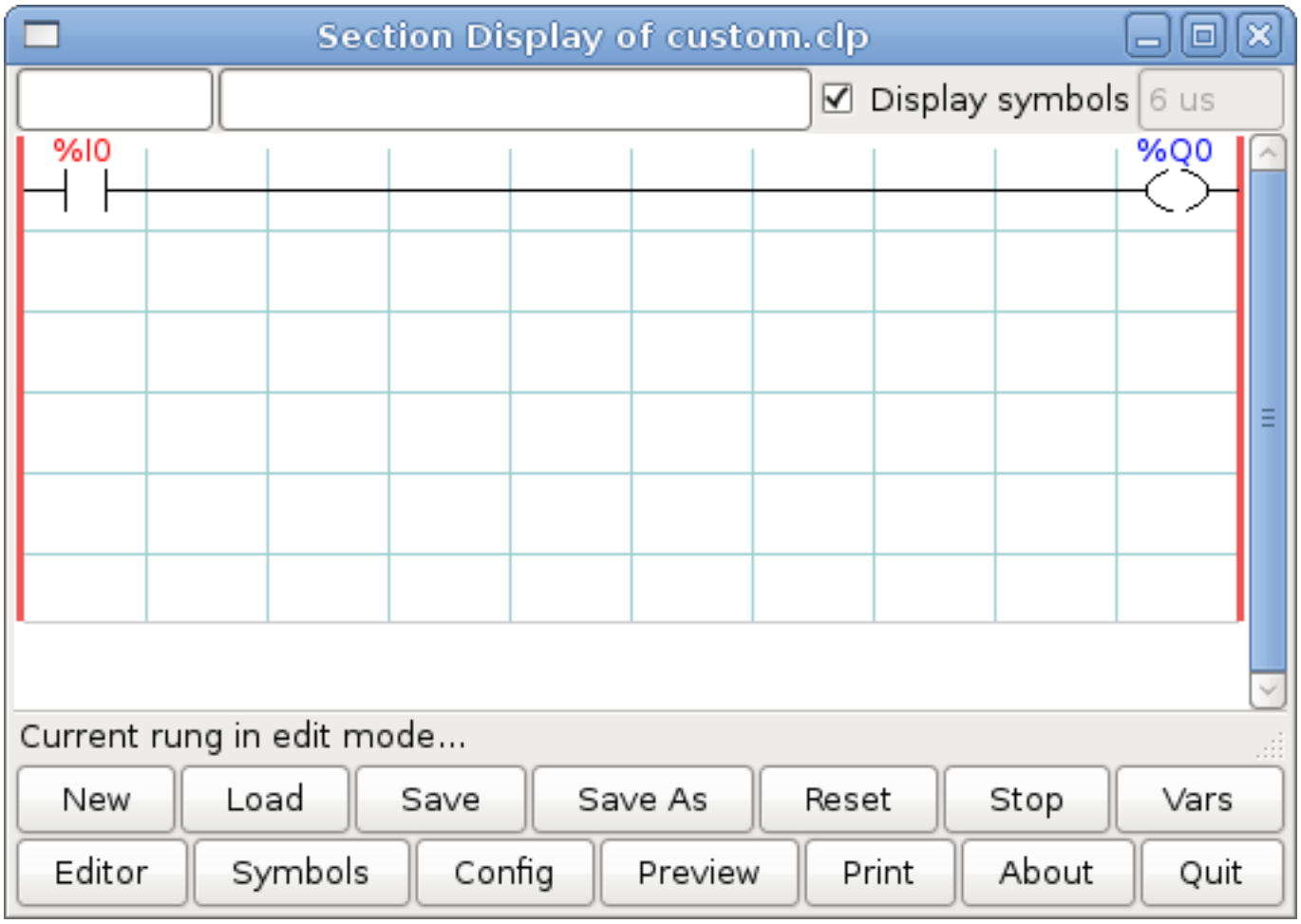


Figure 34.15: Section Display with Rung

Now click on the OK button in the Editor window. Now your Section Display should look like this.

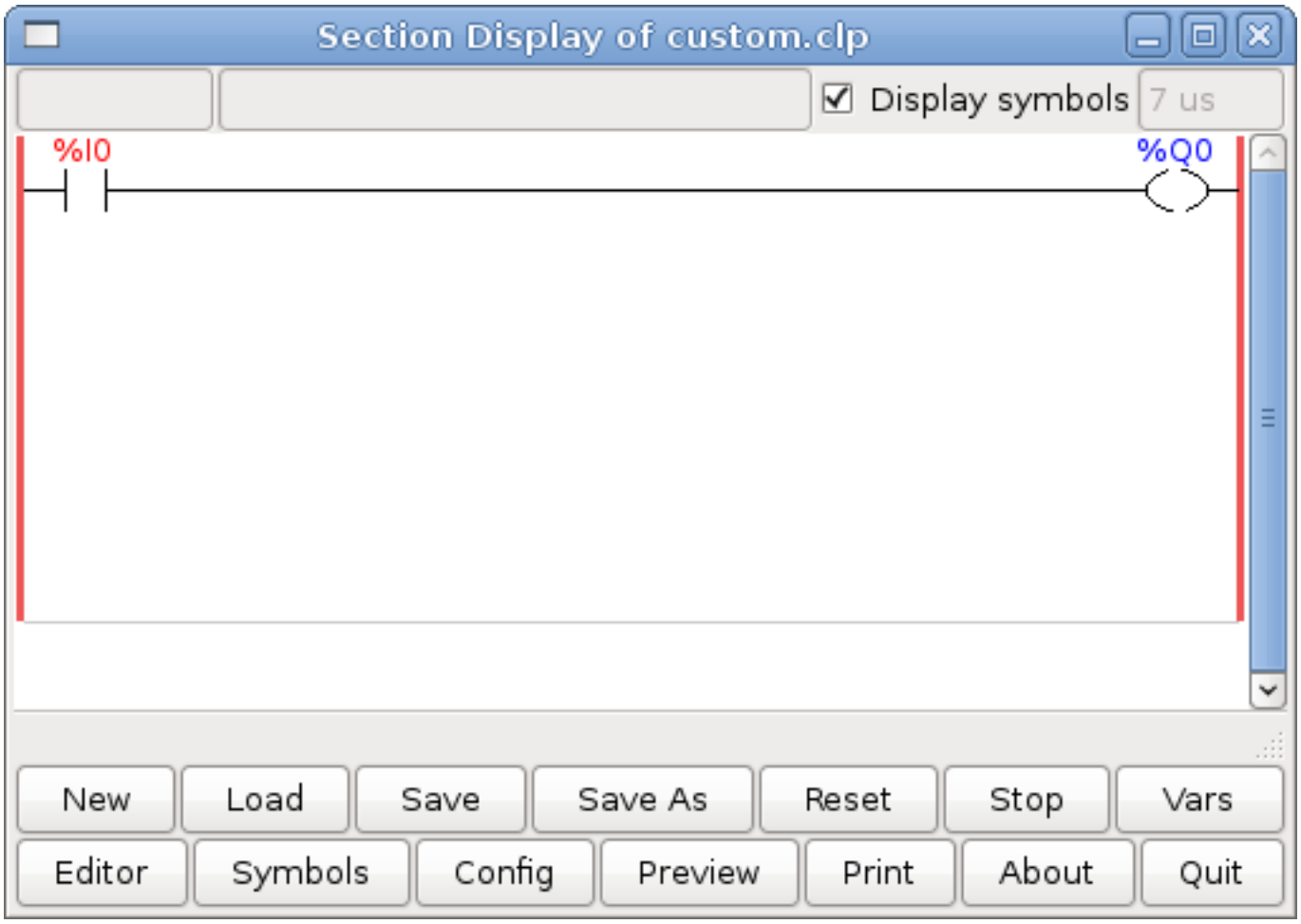


Figure 34.16: Section Display Finished

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.

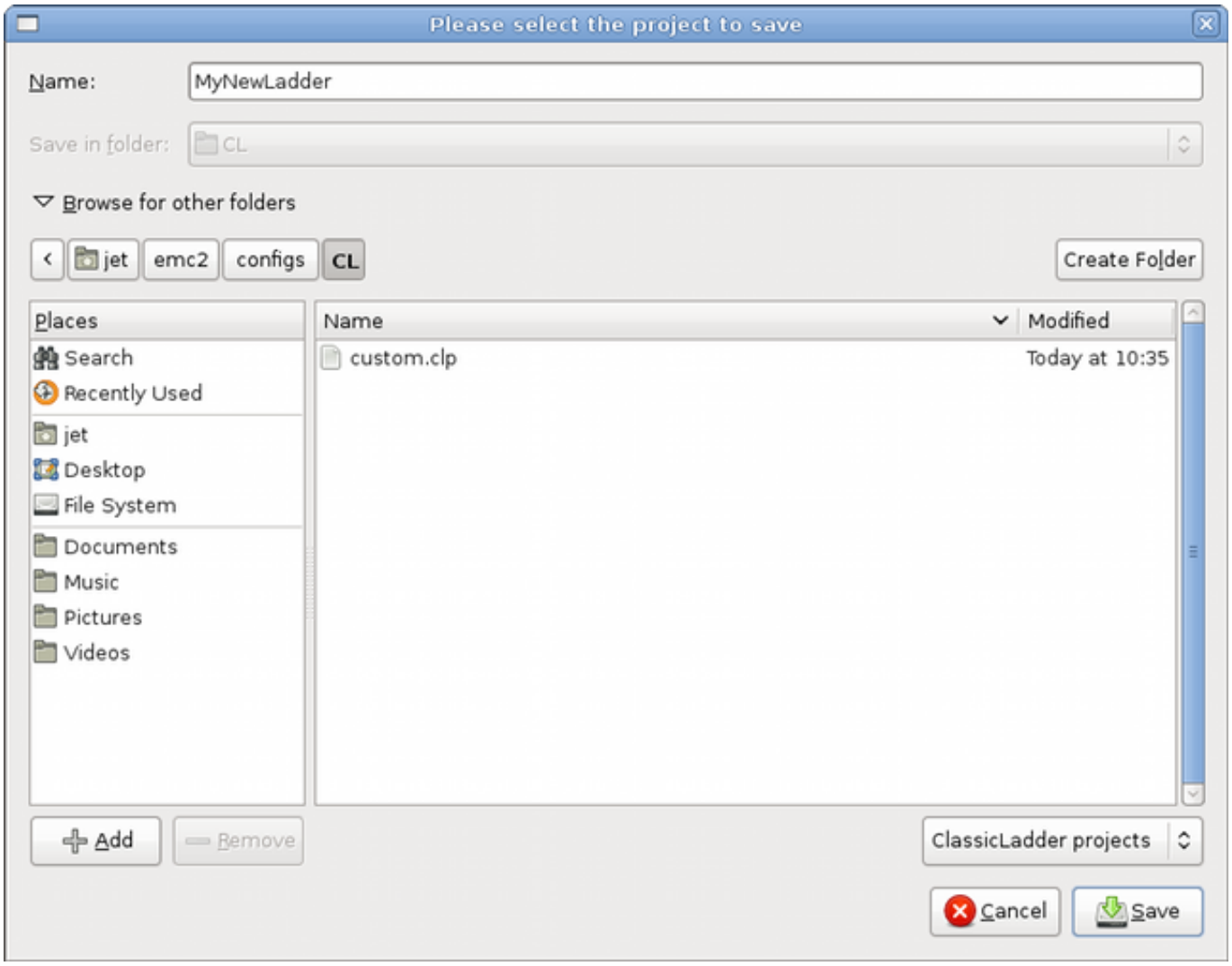


Figure 34.17: Save As Dialog

Again if you used the Stepconf Wizard to add Classic Ladder you can skip this step.

To manually add a ladder you need to add 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.

Chapter 35

Classicladder Examples

35.1 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.

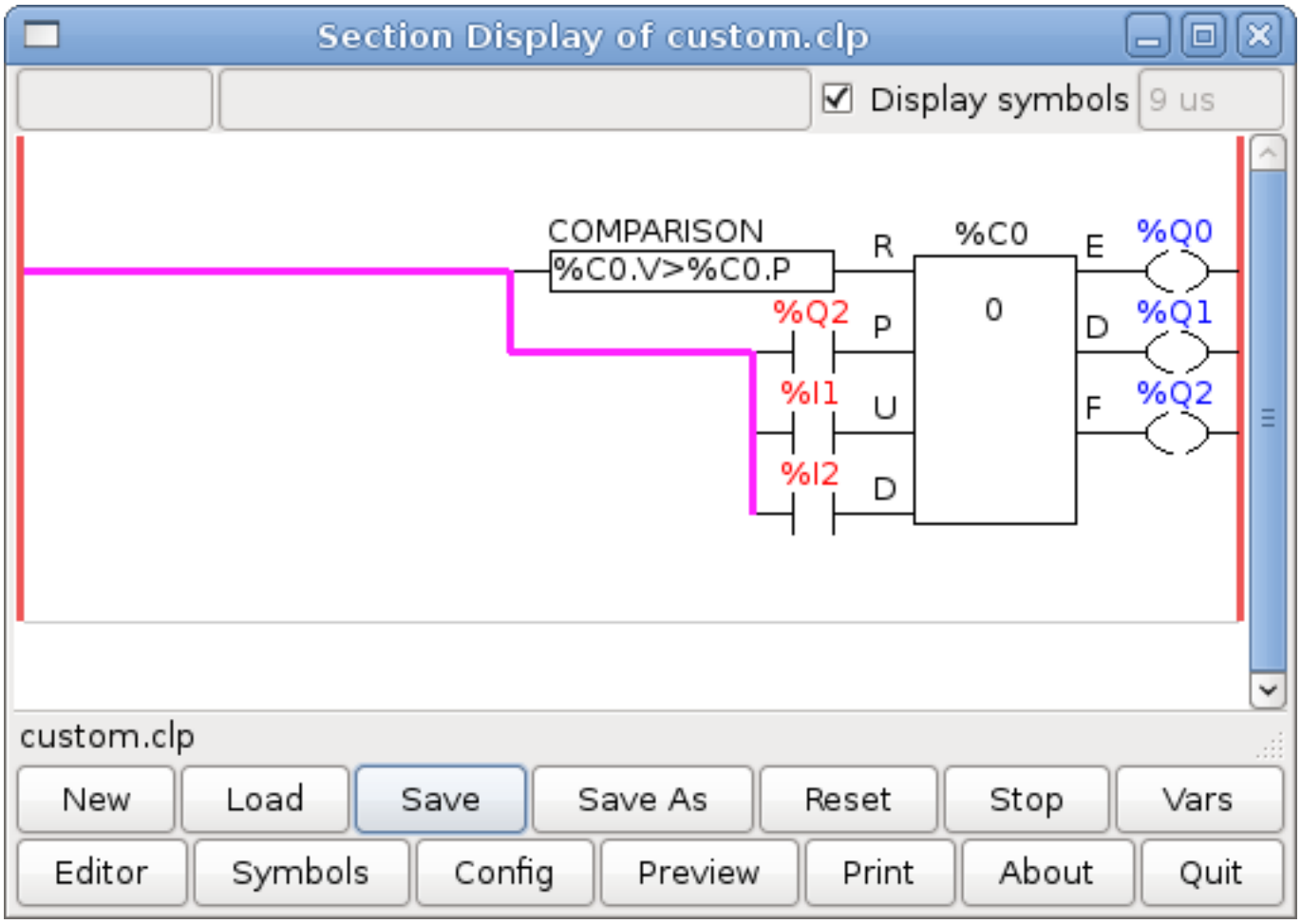


Figure 35.1: Wrapping Counter

35.2 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.

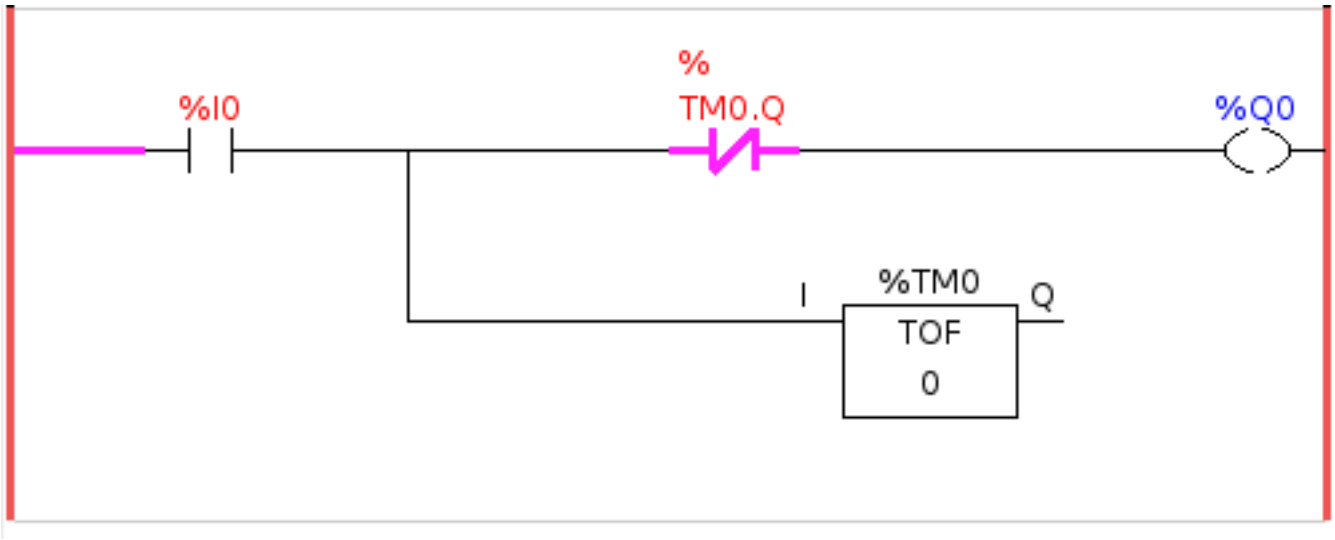


Figure 35.2: Reject Extra Pulse

35.3 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 Classic Ladder.

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 Classic Ladder 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.

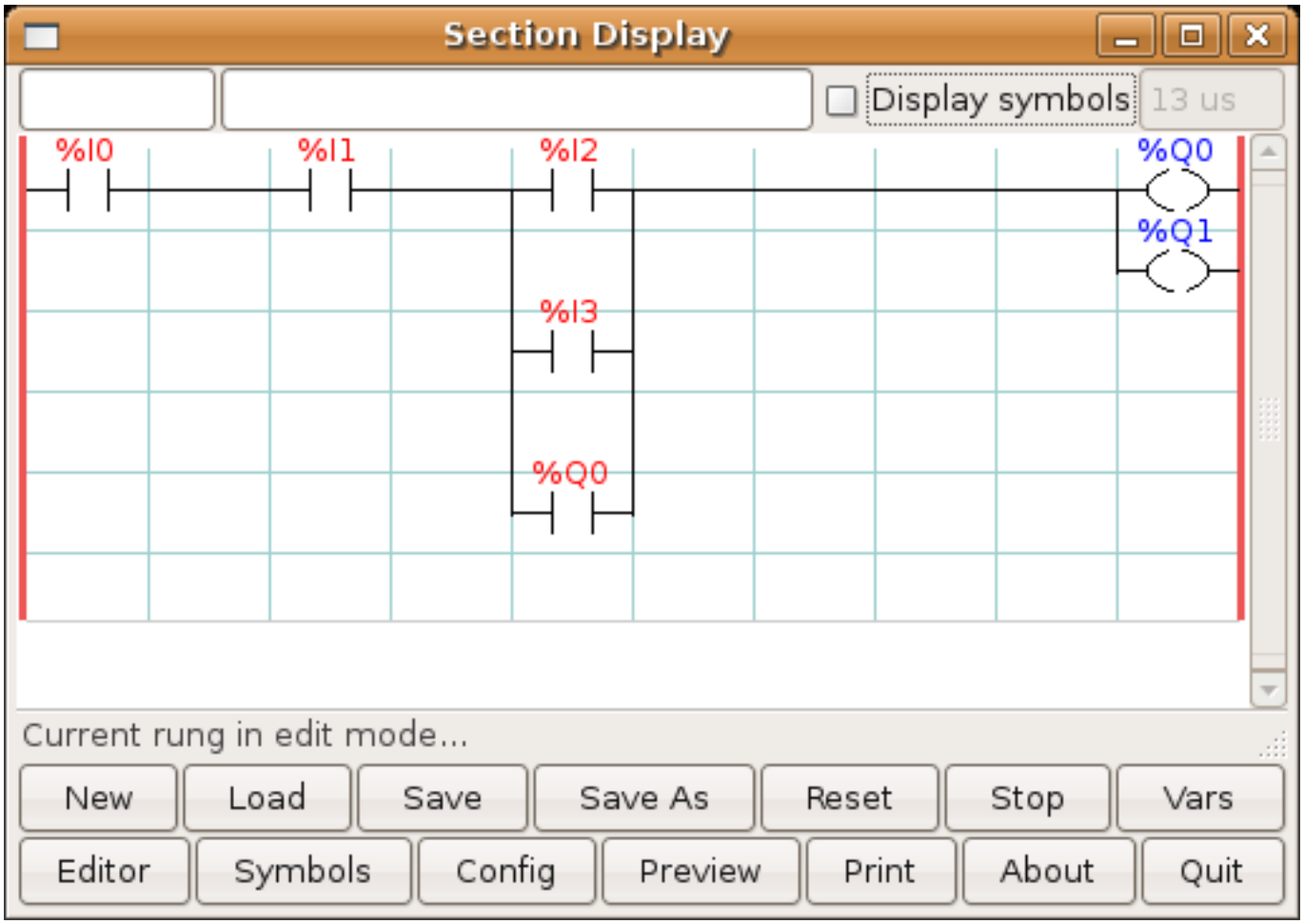


Figure 35.3: E-Stop Section Display

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 = Ouput 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 Classic Ladder
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.

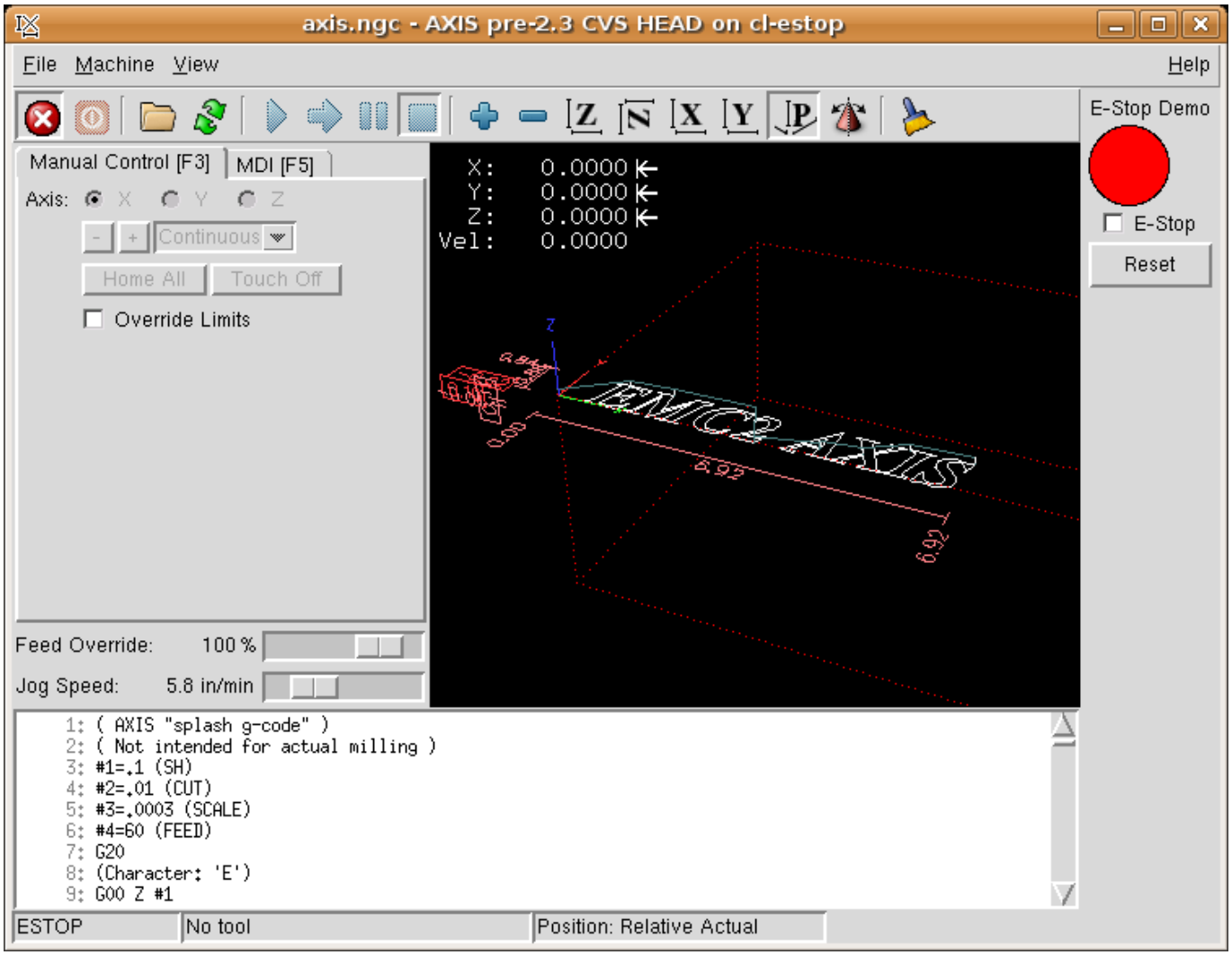


Figure 35.4: AXIS E-Stop

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.

35.4 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.

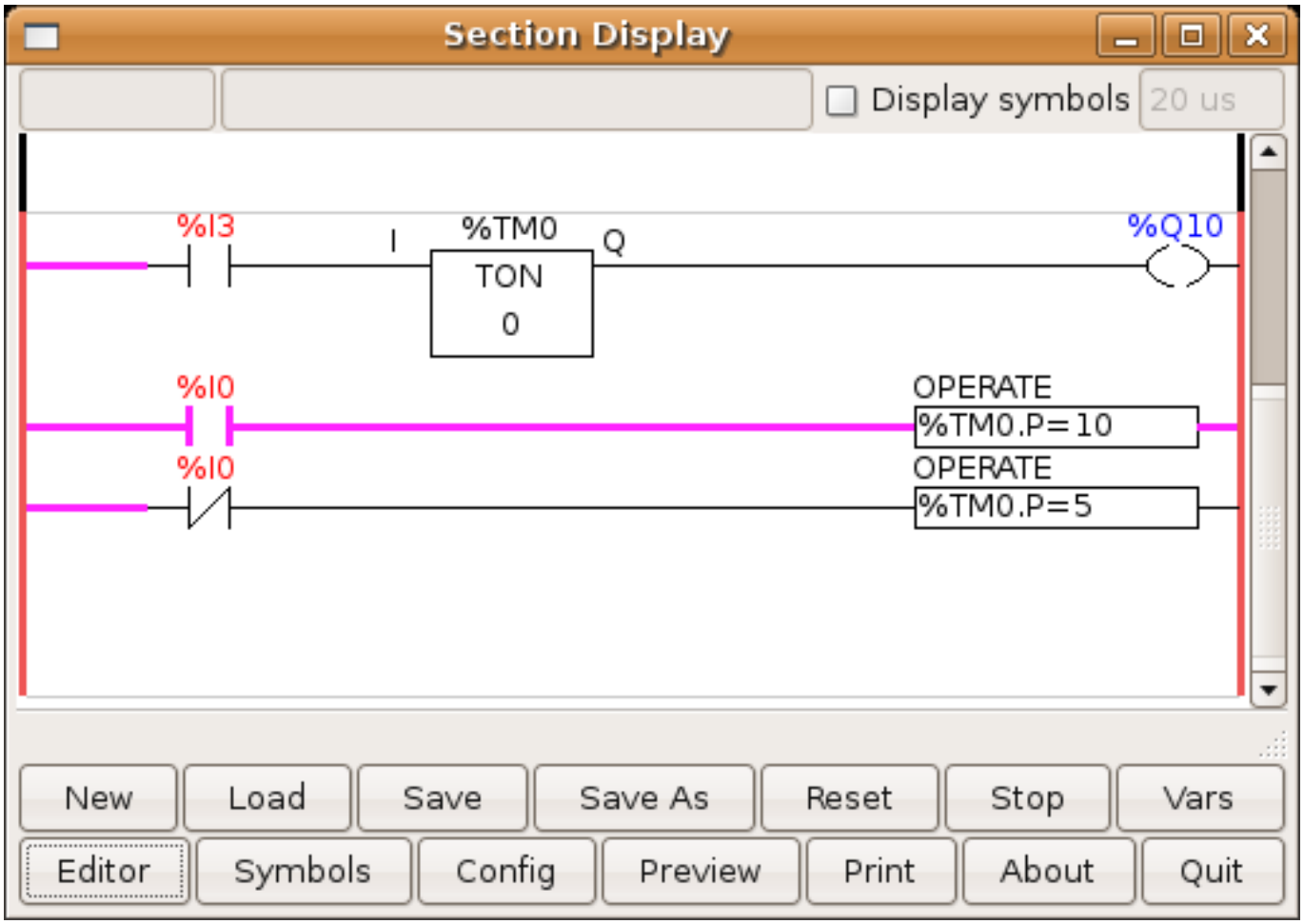


Figure 35.5: Timer/Operate Example

In this case %I0 is true so the timer preset value is 10. If %I0 was false the timer preset would be 5.

Part VII

Hardware Examples

Chapter 36

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.

Chapter 37

Spindle Control

37.1 0-10v Spindle Speed

If your spindle speed is controlled by an analog signal, (for example, by a VFD with a 0 to 10 volt 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. For this example the spindle top speed of 5000 RPM is equal to 10 volts.

$$\frac{10\,Volts}{5000\,RPM} = \frac{0.002\,Volts}{1\,RPM}$$

We have to add a scale component to the HAL file to scale the motion.spindle-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 motion.spindle-speed-out => scale.0.in
net spindle-speed-DAC scale.0.out => <your DAC pin name>
```

37.2 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 motion.spindle-speed-out => pwmgen.0.value
net spindle-on motion.spindle-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.

37.3 Spindle Enable

If you need a spindle enable signal, link your output pin to motion.spindle-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 motion.spindle-on => parport.0.pin-14-out
```

37.4 Spindle Direction

If you have direction control of your spindle the HAL pins motion.spindle-forward and motion.spindle-reverse are controlled by 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 motion.spindle-forward => parport.0.pin-16-out
net spindle-rev motion.spindle-reverse => parport.0.pin-17-out
```

37.5 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 motion.spindle-speed-out and run it through a limit2 component with the scale set so it will ramp the rpm from motion.spindle-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 motion.spindle-speed-out => scale.0.in* is changed as shown in the following example:

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 real time a limit2 and a near with names so it is easier to follow
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 <= motion.spindle-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 motion.spindle-at-speed
# and when this is true motion will start
net spindle-ready <= spindle-at-speed.out => motion.spindle-at-speed
```

37.6 Spindle Feedback

37.6.1 Spindle Synchronized Motion

Spindle feedback is needed by LinuxCNC to perform any spindle coordinated motions like threading and constant surface speed. The StepConf Wizard can perform the connections for you if you select Encoder Phase A and Encoder Index as inputs.

Hardware assumptions:

- 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=1
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 => motion.spindle-revs
net spindle-velocity encoder.3.velocity => motion.spindle-speed-in
net spindle-index-enable encoder.3.index-enable <=> motion.spindle-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
```

[1] In 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.

[2] The HAL encoder index-enable is an exception to the rule in that it behaves as both an input and an output, see manual for details

[3] It is because we selected *non-quadrature simple counting…* above that we can get away with *quadrature* counting without having any B quadrature input.

37.6.2 Spindle At Speed

To enable LinuxCNC to wait for the spindle to be at speed before executing a series of moves you need to set motion.spindle-at-speed to true when the spindle is at the commanded speed. To do this you need spindle feedback from an encoder. Since the feedback and the commanded speed are not usually *exactly* the same you need to use the *near* component to say 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 motion.spindle-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 motion.spindle-at-speed <= near.0.out

# set the spindle speed inputs to agree if within 1%
setp near.0.scale 1.01
```

Chapter 38

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.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.0.jog-vel-mode 0
setp axis.1.jog-vel-mode 0
setp axis.2.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.0.jog-scale
```

```
net mpg-scale => axis.1.jog-scale
net mpg-scale => axis.2.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.0.jog-enable <= parport.1.pin-04-in
net mpg-y axis.1.jog-enable <= parport.1.pin-05-in
net mpg-z axis.2.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.0.jog-counts
net encoder-counts => axis.1.jog-counts
net encoder-counts => axis.2.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.0.jog-vel-mode 0
setp axis.1.jog-vel-mode 0
setp axis.2.jog-vel-mode 0

# 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

# 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 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.0.jog-scale
net mpg-scale => axis.1.jog-scale
net mpg-scale => axis.2.jog-scale

# 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.0.jog-counts
net encoder-counts => axis.1.jog-counts
net encoder-counts => axis.2.jog-counts
```

Chapter 39

GS2 Spindle

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 <= motion.spindle-forward

# connect the spindle on pin to the GS2
net gs2-run spindle-vfd.spindle-on <= motion.spindle-on

# connect the GS2 at speed to the motion at speed
net gs2-at-speed motion.spindle-at-speed <= spindle-vfd.at-speed

# connect the spindle RPM to the GS2
net gs2-RPM spindle-vfd.speed-command <= motion.spindle-speed-out
```

Note
The transmission speed might be able to be faster depending on the exact envirnment. 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.

Part VIII

Diagnostics

Chapter 40

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.

40.1 Common Problems

40.1.1 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.

40.1.2 No Steppers Move

Many drives have an enable pin or need a charge pump to enable the output.

40.1.3 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.

40.2 Error Messages

40.2.1 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.

40.2.2 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.

40.3 Testing

40.3.1 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 41

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 comtrol 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 controled with hardware. Gmoccapy is highly cusomizable.

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, gcode 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 gcode program to properly fit the true location of the vise 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.

Chapter 42

Legal Section

42.1 Copyright Terms

42.2 GNU Free Documentation License

GNU Free Documentation License Version 1.1, March 2000

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. Secondarily, 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 43

Index

_
.linuxcncrc, 14
0-10v Spindle Speed, 325

A
ACEX1K, 226
acme screw, 337
addf, 56
ANGULAR UNITS, 27
APPLICATIONS (inifile section), 24
AX5214H Driver, 201
axis, 19, 337
axis (HAL pins), 47
AXIS (inifile section), 27
axis-iterface, 241
axis-pinout, 240

B
Backlash, 28
backlash, 337
backlash compensation, 337
ball nut, 337
ball screw, 337
Basic HAL Reference, 55
Bit, 60

C
can-parameters, 250
can-pins, 250
Cartesian machines, 272
Classicladder Examples, 316
Classicladder Introduction, 283
Classicladder Programming, 286
CNC, 338
Comments
    INI File, 16
comp, 338
Compensation, 28
connecting-rs485, 254
coordinate measuring machine, 338
Core Components, 43

D
dac-parameters, 248
dac-pins, 248
DISPLAY (inifile section), 19
display units, 338
DRO, 338

E
e-stop-switch-parameters, 252
EDM, 338
EMC, 338
EMC (inifile section), 19
EMCIO, 338
EMCIO (inifile section), 32
EMCMOT, 338
EMCMOT (inifile section), 23
enable signal, 53
enable-pins, 247
Enabling optional features, 101
encoder, 31, 32, 338
encoder-parameters, 242
encoder-pins, 242
end-and-homing-switch-connector-pinout, 251
end-and-homing-switch-pins, 252
ESTOP, 53

F
feed, 338
feed rate, 338
feedback, 338
feedrate override, 339
FERROR, 28
Float, 60

G
G-Code, 339
General Mechatronics Driver, 237
Glade Virtual Control Panel, 138
gmoccapy, 19
gpio-parameters, 239
gpio-pinout, 239
gpio-pins, 239
GS2 Spindle, 332
GS2 VFD Driver, 203
gscreen, 19

GUI, 337, 339

H
HAL, 13, 339
HAL (inifile section), 23
HAL User Interface, 187
HALTCL Files, 40
HALUI (inifile section), 24
HOME, 37
home, 339
HOME IGNORE LIMITS, 36
HOME IS SHARED, 37
HOME LATCH VEL, 36
HOME OFFSET, 37
HOME SEARCH VEL, 29, 36
HOME SEQUENCE, 37
HOME USE INDEX, 37
Homing Configuration, 34

I
Immediate Homing, 38
INI, 13, 339
ini [FILTER] Section, 21
INI Configuration, 16
INI File, 16
ini settings (HAL pins), 49
Instance, 339
Integrator Concepts, 3
iocontrol (HAL pins), 48

J
jog, 339
joint coordinates, 339

K
keystick, 19
Kinematics, 272
kinematics, 272, 339

L
Latency Test, 8
Lathe Configuration, 39
lead screw, 339
LINEAR UNITS, 26
loadrt, 56
loadusr, 57
LOCKING INDEXER, 37
loop, 340

M
machine on, 54
machine units, 339
MAX ACCELERATION, 27
MAX LIMIT, 28
MAX VELOCITY, 27
MDI, 190, 340
Mesa HostMot2 Driver, 205
MIN FERROR, 28
MIN LIMIT, 28
mini, 19
Motenc Driver, 217
motion (HAL pins), 44

N
net, 57
NIST, 340
NML, 13, 340

O
offsets, 340
Opto22 Driver, 219
ORIENT OFFSET, 22

P
Parallel Port Driver, 196
PARAMETER FILE, 22
part Program, 340
PCI Parallel Port, 324
pci-card connectors, 238
Pico PPMC Driver, 222
pin-numbering-axis, 240
pin-numbering-endsw, 251
pin-numbering-gpio, 238
pinout, 50
Pluto P Driver, 226
pluto-servo, 227
pluto-servo alternate pin functions, 229
pluto-servo pinout, 228
pluto-step, 230
pluto-step pinout, 231
pluto-step timings, 232
program units, 340
PWM Spindle Speed, 325
Python Interface, 260
Python Virtual Control Panel, 108

R
rapid, 340
rapid rate, 340
real-time, 340
refsig-timing-diagram, 246
RS274NGC, 340
RS274NGC (inifile section), 22
RS274NGC STARTUP CODE, 22
rs485-dacadc-parameters, 256
rs485-dacadc-pins, 256
rs485-input-pins, 255
rs485-relay-parameters, 255
rs485-relay-pins, 254
rs485-teachpendant-parameters, 257
rs485-teachpendant-pins, 257
RTAI, 340
RTAPI, 340
RTLINUX, 340

S

s32, 60
servo motor, 340
Servo To Go Driver, 233
setp, 58
sets, 59
ShuttleXpress, 235
signal polarity, 53
Signed Integer, 341
spindle, 341
Spindle At Speed, 328
Spindle Control, 325
Spindle Direction, 326
Spindle Enable, 326
Spindle Feedback, 327
Spindle Soft Start, 326
spindle speed control, 53
Spindle Synchronized Motion, 327
standard pinout, 51
Starting LinuxCNC, 12
step rate, 50
stepgen-parameters, 245
stepgen-pins, 245
stepper, 50
Stepper Configuration, 50
Stepper Diagnostics, 334
stepper motor, 341
Stepper Tuning, 276
SUBROUTINE PATH, 22

T
TASK, 341
TASK (inifile section), 23
TBL, 13
time, 61
Tk, 341
tkLinuxCNC, 19
tmax, 61
touchy, 19
TRAJ (inifile section), 25
Traverse Move, 341
Trivial Kinematics, 272

U
u32, 60
UNITS, 28
units, 341
Unsigned Integer, 341
USER M PATH, 22

V
VAR, 13
VOLATILE HOME, 37

W
watchdog-parameters, 251
watchdog-pins, 250
world coordinates, 341

X
xemc, 19