**Glade Virtual Control Panel**

Table of Contents

[1. What is GladeVCP?](http://www.linuxcnc.org/docs/2.5/html/gui/gladevcp.html#_what_is_gladevcp)

[1.1. PyVCP versus GladeVCP at a glance](http://www.linuxcnc.org/docs/2.5/html/gui/gladevcp.html#_pyvcp_versus_gladevcp_at_a_glance)

[2. A Quick Tour with the Example Panel](http://www.linuxcnc.org/docs/2.5/html/gui/gladevcp.html#_a_quick_tour_with_the_example_panel)

[2.1. Exploring the example panel](http://www.linuxcnc.org/docs/2.5/html/gui/gladevcp.html#_exploring_the_example_panel)

[2.2. Exploring the User Interface description](http://www.linuxcnc.org/docs/2.5/html/gui/gladevcp.html#_exploring_the_user_interface_description)

[2.3. Exploring the Python callback](http://www.linuxcnc.org/docs/2.5/html/gui/gladevcp.html#_exploring_the_python_callback)

[3. Creating and Integrating a Glade user interface](http://www.linuxcnc.org/docs/2.5/html/gui/gladevcp.html#_creating_and_integrating_a_glade_user_interface)

[3.1. Prerequisite: Glade installation](http://www.linuxcnc.org/docs/2.5/html/gui/gladevcp.html#gladevcp:Prerequisites)

[3.2. Running Glade to create a new user interface](http://www.linuxcnc.org/docs/2.5/html/gui/gladevcp.html#_running_glade_to_create_a_new_user_interface)

[3.3. Testing a panel](http://www.linuxcnc.org/docs/2.5/html/gui/gladevcp.html#_testing_a_panel)

[3.4. Preparing the HAL command file](http://www.linuxcnc.org/docs/2.5/html/gui/gladevcp.html#_preparing_the_hal_command_file)

[3.5. Integrating into Axis like PyVCP](http://www.linuxcnc.org/docs/2.5/html/gui/gladevcp.html#_integrating_into_axis_like_pyvcp)

[3.6. Integrating into Axis as a tab next to DRO and Preview](http://www.linuxcnc.org/docs/2.5/html/gui/gladevcp.html#_integrating_into_axis_as_a_tab_next_to_dro_and_preview)

[3.7. Integrating into Touchy](http://www.linuxcnc.org/docs/2.5/html/gui/gladevcp.html#_integrating_into_touchy)

[4. GladeVCP command line options](http://www.linuxcnc.org/docs/2.5/html/gui/gladevcp.html#_gladevcp_command_line_options)

[5. Understanding the gladeVCP startup process](http://www.linuxcnc.org/docs/2.5/html/gui/gladevcp.html#_understanding_the_gladevcp_startup_process)

[6. HAL Widget reference](http://www.linuxcnc.org/docs/2.5/html/gui/gladevcp.html#_hal_widget_reference)

[6.1. Widget and HAL pin naming](http://www.linuxcnc.org/docs/2.5/html/gui/gladevcp.html#_widget_and_hal_pin_naming)

[6.2. Python attributes and methods of HAL Widgets](http://www.linuxcnc.org/docs/2.5/html/gui/gladevcp.html#_python_attributes_and_methods_of_hal_widgets)

[6.3. Setting pin and widget values](http://www.linuxcnc.org/docs/2.5/html/gui/gladevcp.html#_setting_pin_and_widget_values)

[6.4. The hal-pin-changed signal](http://www.linuxcnc.org/docs/2.5/html/gui/gladevcp.html#gladevcp::hal-pin-changed_signal)

[6.5. Buttons](http://www.linuxcnc.org/docs/2.5/html/gui/gladevcp.html#gladevcp:HAL_Buttons)

[6.6. Scales](http://www.linuxcnc.org/docs/2.5/html/gui/gladevcp.html#gladevcp:HAL_Scales)

[6.7. SpinButton](http://www.linuxcnc.org/docs/2.5/html/gui/gladevcp.html#gladevcp:HAL_SpinButton)

[6.8. Label](http://www.linuxcnc.org/docs/2.5/html/gui/gladevcp.html#gladevcp:HAL_Label)

[6.9. Containers: HAL\_HBox and HAL\_Table](http://www.linuxcnc.org/docs/2.5/html/gui/gladevcp.html#gladevcp:HAL_HBox_Table)

[6.10. LED](http://www.linuxcnc.org/docs/2.5/html/gui/gladevcp.html#gladevcp:HAL_LED)

[6.11. ProgressBar](http://www.linuxcnc.org/docs/2.5/html/gui/gladevcp.html#gladevcp:HAL_ProgressBar)

[6.12. ComboBox](http://www.linuxcnc.org/docs/2.5/html/gui/gladevcp.html#gladevcp:HAL_ComboBox)

[6.13. Bars](http://www.linuxcnc.org/docs/2.5/html/gui/gladevcp.html#gladevcp:HAL_Bars)

[6.14. Meter](http://www.linuxcnc.org/docs/2.5/html/gui/gladevcp.html#gladevcp:HAL_Meter)

[6.15. Gremlin tool path preview for .ngc files](http://www.linuxcnc.org/docs/2.5/html/gui/gladevcp.html#gladevcp:HAL_Gremlin)

[6.16. Animated function diagrams: HAL widgets in a bitmap](http://www.linuxcnc.org/docs/2.5/html/gui/gladevcp.html#_animated_function_diagrams_hal_widgets_in_a_bitmap)

[7. Action Widgets reference](http://www.linuxcnc.org/docs/2.5/html/gui/gladevcp.html#_action_widgets_reference)

[7.1. EMC Action widgets](http://www.linuxcnc.org/docs/2.5/html/gui/gladevcp.html#_emc_action_widgets)

[7.2. EMC ToggleAction widgets](http://www.linuxcnc.org/docs/2.5/html/gui/gladevcp.html#_emc_toggleaction_widgets)

[7.3. The Action\_MDI Toggle and Action\_MDI widgets](http://www.linuxcnc.org/docs/2.5/html/gui/gladevcp.html#_the_action_mdi_toggle_and_action_mdi_widgets)

[7.4. A simple example: Execute MDI command on button press](http://www.linuxcnc.org/docs/2.5/html/gui/gladevcp.html#_a_simple_example_execute_mdi_command_on_button_press)

[7.5. Parameter passing with Action\_MDI and ToggleAction\_MDI widgets](http://www.linuxcnc.org/docs/2.5/html/gui/gladevcp.html#_parameter_passing_with_action_mdi_and_toggleaction_mdi_widgets)

[7.6. An advanced example: Feeding parameters to an O-word subroutine](http://www.linuxcnc.org/docs/2.5/html/gui/gladevcp.html#_an_advanced_example_feeding_parameters_to_an_o_word_subroutine)

[7.7. Preparing for an MDI Action, and cleaning up afterwards](http://www.linuxcnc.org/docs/2.5/html/gui/gladevcp.html#_preparing_for_an_mdi_action_and_cleaning_up_afterwards)

[7.8. Using the LinuxCNC Stat object to deal with status changes](http://www.linuxcnc.org/docs/2.5/html/gui/gladevcp.html#_using_the_linuxcnc_stat_object_to_deal_with_status_changes)

[8. GladeVCP Programming](http://www.linuxcnc.org/docs/2.5/html/gui/gladevcp.html#gladevcp:GladeVCP_Programming)

[8.1. User Defined Actions](http://www.linuxcnc.org/docs/2.5/html/gui/gladevcp.html#_user_defined_actions)

[8.2. An example: adding custom user callbacks in Python](http://www.linuxcnc.org/docs/2.5/html/gui/gladevcp.html#_an_example_adding_custom_user_callbacks_in_python)

[8.3. HAL value change events](http://www.linuxcnc.org/docs/2.5/html/gui/gladevcp.html#_hal_value_change_events)

[8.4. Programming model](http://www.linuxcnc.org/docs/2.5/html/gui/gladevcp.html#_programming_model)

[8.5. Initialization sequence](http://www.linuxcnc.org/docs/2.5/html/gui/gladevcp.html#_initialization_sequence)

[8.6. Multiple callbacks with the same name](http://www.linuxcnc.org/docs/2.5/html/gui/gladevcp.html#_multiple_callbacks_with_the_same_name)

[8.7. The GladeVCP -U <useropts> flag](http://www.linuxcnc.org/docs/2.5/html/gui/gladevcp.html#_the_gladevcp_tt_u_lt_useropts_gt_tt_flag)

[8.8. Persistent variables in GladeVCP](http://www.linuxcnc.org/docs/2.5/html/gui/gladevcp.html#_persistent_variables_in_gladevcp)

[8.9. Using persistent variables](http://www.linuxcnc.org/docs/2.5/html/gui/gladevcp.html#_using_persistent_variables)

[8.10. Saving the state on Gladvcp shutdown](http://www.linuxcnc.org/docs/2.5/html/gui/gladevcp.html#_saving_the_state_on_gladvcp_shutdown)

[8.11. Saving state when Ctrl-C is pressed](http://www.linuxcnc.org/docs/2.5/html/gui/gladevcp.html#_saving_state_when_ctrl_c_is_pressed)

[8.12. Hand-editing .ini files](http://www.linuxcnc.org/docs/2.5/html/gui/gladevcp.html#_hand_editing_ini_files)

[8.13. Adding HAL pins](http://www.linuxcnc.org/docs/2.5/html/gui/gladevcp.html#gladevcp:Adding_HAL_pins)

[8.14. Adding timers](http://www.linuxcnc.org/docs/2.5/html/gui/gladevcp.html#_adding_timers)

[8.15. Setting HAL widget properties programmatically](http://www.linuxcnc.org/docs/2.5/html/gui/gladevcp.html#_setting_hal_widget_properties_programmatically)

[8.16. Examples, and rolling your own GladeVCP application](http://www.linuxcnc.org/docs/2.5/html/gui/gladevcp.html#_examples_and_rolling_your_own_gladevcp_application)

[9. FAQ](http://www.linuxcnc.org/docs/2.5/html/gui/gladevcp.html#_faq)

[10. Troubleshooting](http://www.linuxcnc.org/docs/2.5/html/gui/gladevcp.html#_troubleshooting)

[11. Implementation note: Key handling in Axis](http://www.linuxcnc.org/docs/2.5/html/gui/gladevcp.html#_implementation_note_key_handling_in_axis)

[12. Adding Custom Widgets](http://www.linuxcnc.org/docs/2.5/html/gui/gladevcp.html#_adding_custom_widgets)

**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 or Touchy. Unlike PyVCP, GladeVCP is not limitied 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 users the [Glade](http://glade.gnome.org/) WYSIWYG user interface editor, which makes it easy to create visually pleasing panels. It relies on the [PyGTK](http://www.pygtk.org/) bindings to the rich [GTK+](http://www.gtk.org/) 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.

**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+](http://www.gtk.org/) widget set.
* user interface creation: uses the [Glade](http://glade.gnome.org/) 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

**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 and Touchy; 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](http://www.linuxcnc.org/docs/2.5/html/common/starting-emc.html#cha:starting-linuxcnc) in the *Sample Configurations > sim > gladevcp* branch.

Git Checkout

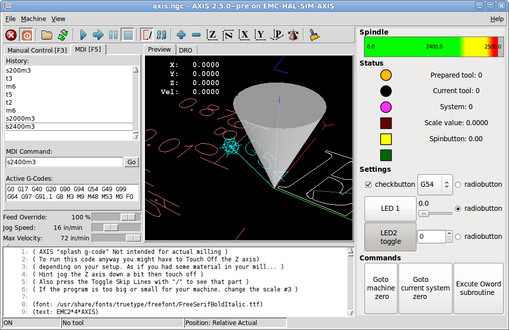
The following instructions only apply if you’re using a git checkout. Open a terminal and change to the directory created by git then issue the commmands as shown.

|  |  |
| --- | --- |
| 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/gladevcp

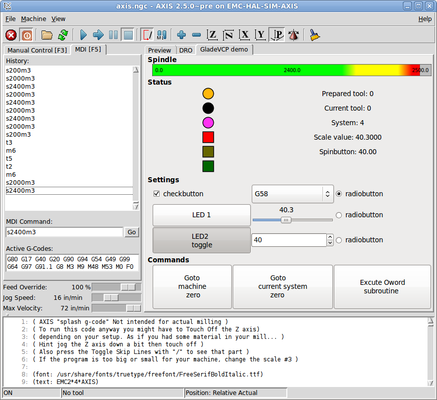
$ linuxcnc gladevcp\_panel.ini



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

$ cd configs/sim/gladevcp

$ linuxcnc gladevcp\_tab.ini

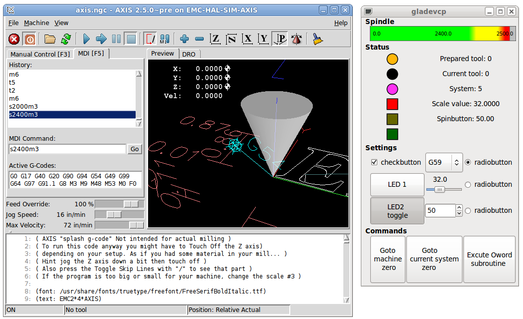


To run this panel as a standalone toplevel window besides Axis, just start Axis in the background and start gladevcp as follows:

$ cd configs/sim/axis

$ linuxcnc axis.ini &

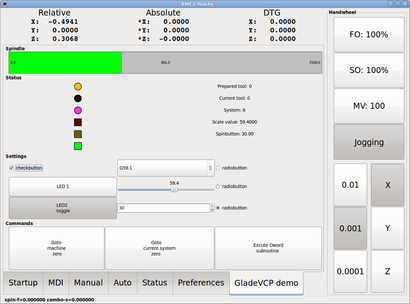
$ gladevcp -c gladevcp -u ../gladevcp/hitcounter.py -H ../gladevcp/manual-example.hal ../gladevcp/manual-example.ui



To run this panel inside *Touchy*:

$ cd configs/sim/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.

**2.1. Exploring the example panel**

While running Axis, 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/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 *configs/gladevcp/nc\_files/oword.ngc* for the subroutine body.

To see how the panel is integrated into Axis, see the *[DISPLAY]GLADEVCP* statements in gladevcp\_panel.ui, and the *[DISPLAY]EMBED\** and *[HAL]POSTGUI\_HALFILE* statements in gladevcp\_tab.ini respectively.

**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](http://www.linuxcnc.org/docs/2.5/html/gui/gladevcp.html#gladevcp:Prerequisites). To edit the user interface, run the command

$ glade configs/gladevcp/manual-example.ui

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, unter the *General* tab, the *MDI command* property.

**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 ../gladevcp/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](http://www.linuxcnc.org/docs/2.5/html/gui/gladevcp.html#gladevcp:GladeVCP_Programming) section.

**3. Creating and Integrating a Glade user interface**

**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

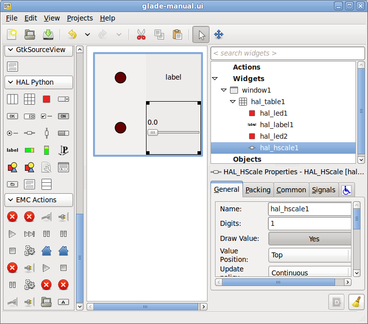
**3.2. Running Glade to create a new user interface**

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](http://www.youtube.com).

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*.

**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.

**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. |

**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 ../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 = ../gladevcp/nc\_files/

The HAL component name of a GladeVCP application started with the 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 [RS274NGC]SUBROUTINE\_PATH= option is only set so the example panel will find the Oword subroutine for the MDI Command widget. It might not be needed in your setup. |

**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 = ../gladevcp/nc\_files/

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. |

**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 ../gladevcp/hitcounter.py -H ../gladevcp/gladevcp-touchy.hal ../gladevcp/manual-example.ui

**[RS274NGC]**

*# gladevcp Demo specific Oword subs live here*

SUBROUTINE\_PATH = ../gladevcp/nc\_files/

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.

**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](http://wiki.linuxcnc.org/cgi-bin/wiki.pl?GTK_Themes).

-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

**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 (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).

**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](http://www.linuxcnc.org/docs/2.5/html/hal/basic_hal.html#sec:Hal-Data). 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](http://www.linuxcnc.org/docs/2.5/html/gui/gladevcp.html#gladevcp:HAL_LED)
* Widgets for controlling "bit" signals: [HAL\_Button HAL\_RadioButton HAL\_CheckButton](http://www.linuxcnc.org/docs/2.5/html/gui/gladevcp.html#gladevcp:HAL_Buttons)
* Widgets for indicating "number" signals: [HAL\_Label](http://www.linuxcnc.org/docs/2.5/html/gui/gladevcp.html#gladevcp:HAL_Label), [HAL\_ProgressBar](http://www.linuxcnc.org/docs/2.5/html/gui/gladevcp.html#gladevcp:HAL_ProgressBar), [HAL\_HBar and HAL\_VBar](http://www.linuxcnc.org/docs/2.5/html/gui/gladevcp.html#gladevcp:HAL_Bars), [HAL\_Meter](http://www.linuxcnc.org/docs/2.5/html/gui/gladevcp.html#gladevcp:HAL_Meter)
* Widgets for controlling "number" signals: [HAL\_SpinButton](http://www.linuxcnc.org/docs/2.5/html/gui/gladevcp.html#gladevcp:HAL_SpinButton), [HAL\_HScale and HAL\_VScale](http://www.linuxcnc.org/docs/2.5/html/gui/gladevcp.html#gladevcp:HAL_Scales)
* Helper widgets: [HAL\_HBox and HAL\_Table](http://www.linuxcnc.org/docs/2.5/html/gui/gladevcp.html#gladevcp:HAL_HBox_Table)
* Tool Path preview: [HAL\_Gremlin](http://www.linuxcnc.org/docs/2.5/html/gui/gladevcp.html#gladevcp:HAL_Gremlin)

**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.

**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](http://www.pygtk.org/docs/pygtk/class-gtkbutton.html) in the [PyGtk Reference Manual](http://www.pygtk.org/docs/pygtk).

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.

**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!.

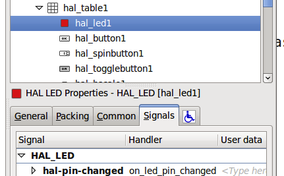
It might be tempting to *set HAL widget input pins* programmatically. Note this defeats the purpose of an input pin in the first place - it should be linked to, and react to signals generated by other HAL components. While there is currently no write protection on writing to input pins in HAL Python, 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).

**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:

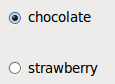
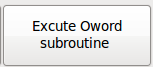


The example in configs/gladevcp/examples/complex shows how this is handled in Python.

**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: images/checkbutton.pngRadio 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/gladevcp/by-widget/radiobutton for a GladeVCP application and UI file for working with radio buttons. |

**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: images/hscale.png.

**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: images/spinbutton.png.

**6.8. 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.

**6.9. Containers: HAL\_HBox and HAL\_Table**

Compared to their Gtk counterparts they have one input BIT 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.

|  |  |
| --- | --- |
| Tip | If you find some part of your GladeVCP application is *grayed out* (insensitive), see whether a container’s pin is unset. |

**6.10. 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.

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: .

**6.11. 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 -224 to +224.

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: images/progressbar2.png.

**6.12. 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/gladevcp/by-widget/combobox/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>.

**6.13. 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.

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.

bg\_color

Background (inactive) color of bar.

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: images/hal_hbar.pngVertical bar: images/vscale.png.

**6.14. 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

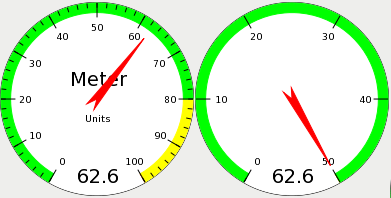
Background color of meter.

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: .

**6.15. 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.

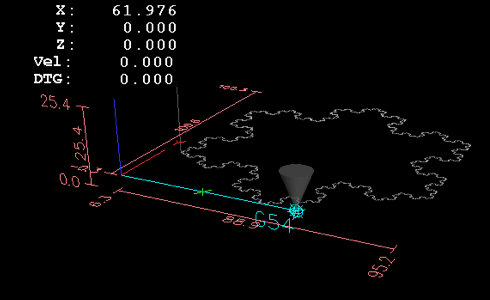
Gremlin does not export any HAL pins. It has the following properties:

view

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

enable\_dro

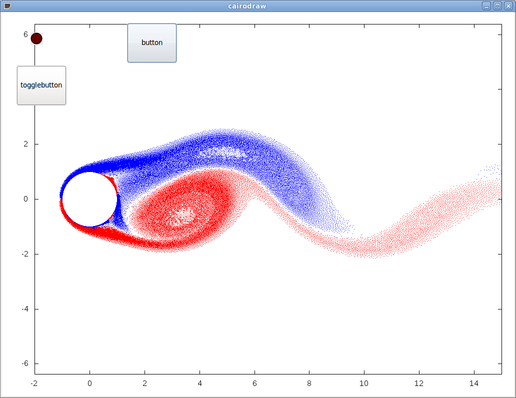
boolean; whether to draw a DRO on the plot or not. Defaults to True.

Example: 

**6.16. 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/gladevcp/animated-backdrop:



**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.

**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.

**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.

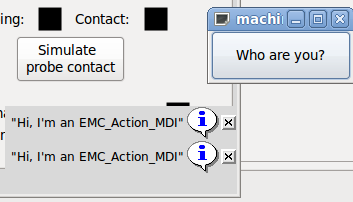
**7.3. The Action\_MDI Toggle and Action\_MDI widgets**

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.

**7.4. A simple example: Execute MDI command on button press**

configs/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:



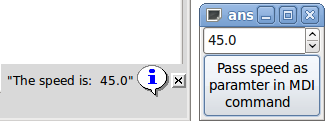
You’ll notice that the button associated with the Action\_MDI action is grayed out if the machine is off, in E-Stop or the interpreter is running. It will automatically become active when the machine is turned on and out of E-Stop, and the program is idle.

**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.

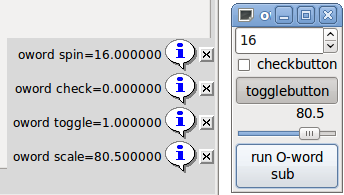
The example UI file is configs/gladevcp/mdi-command-example/speed.ui. Here’s what you get when running it:



**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/gladevcp/mdi-command-example/owordsub.ui.

Place configs/gladevcp/nc\_files/oword.ngc so Axis can find it, and run gladevcp owordsub.ui from a terminal window. This looks like so:



**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. |

**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

**8. GladeVCP Programming**

**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.

**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!

**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](http://www.linuxcnc.org/docs/2.5/html/gui/gladevcp.html#gladevcp:Adding_HAL_pins) section below for details.

HAL widgets come with a pre-defined signal called *hal-pin-changed*. See the [Hal Widgets section](http://www.linuxcnc.org/docs/2.5/html/gui/gladevcp.html#gladevcp::hal-pin-changed_signal) for details.

**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.

**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 un-pythonic. This is why we came up with the class-based handler model.

**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.

**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.

**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.

**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().

**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.

**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.

**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.

**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

**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. |

**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)

**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.

**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**())

**8.14. Adding timers**

Since GladeVCP uses Gtk widgets which rely on the [GObject](http://www.pygtk.org/pygtk2reference/gobject-functions.html) 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)

**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)]

**8.16. Examples, and rolling your own GladeVCP application**

Visit linuxcnc/configs/gladevcp for running examples and starters for your own projects.

**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.

1. *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.

1. *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.

1. *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.

1. *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/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.

1. *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/gladevcp/by-widget/spinbutton.

**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.

**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.

**12. Adding Custom Widgets**

The LinuxCNC Wiki has information on adding custom widgets to GladeVCP. [GladeVCP Custom Widgets](http://wiki.linuxcnc.org/cgi-bin/wiki.pl?GladeVCP_Custom_Widgets)

Last updated 2015-10-24 01:52:21 MDT