

PHENIX Muon Tracking Low-Voltage Control System

Document Revision 1.00

Christopher Smith
Abilene Christian University
bnl@christophersmith.net

July 29, 2004

Contents

1	Introduction	4
2	Operation	4
2.1	Quick Start	4
2.2	Shift Personnel	4
2.3	Muon-Tracking Experts	5
3	Hardware Design	5
3.1	ADAM Network	5
3.2	ADAM Module Layout	6
3.3	Low-Voltage Distribution Cards	6
4	Software Design	6
4.1	Overview	6
4.1.1	Introduction	6
4.1.2	General Notes	7
4.2	GwxLibrary_Main	7
4.2.1	Constants	8
4.2.2	BITMASKS	8

4.2.3	<code>initialize_data()</code>	9
4.2.4	<code>query(module as Integer, slot as Integer)</code>	9
4.2.5	<code>query_group(group as Integer, query_mode as Integer)</code>	9
4.2.6	<code>set_channels(module as Integer, slot as Integer, bitmask as Long, new_state as Boolean)</code>	9
4.2.7	<code>set_group(group as Integer, new_state as Boolean)</code>	9
4.2.8	<code>gcc_startup()</code>	10
4.2.9	<code>gcc_shutdown()</code>	10
4.2.10	<code>set_led_color(r as Integer, g as Integer, b as Integer)</code>	10
4.2.11	<code>wait(seconds as Integer)</code>	11
4.3	<code>GwxButtons_Main</code>	11
4.3.1	<code>camera_startup</code> and <code>camera_shutdown</code>	11
4.3.2	<code>fem_gcc_startup</code>	11
4.3.3	<code>fem_gcc_shutdown</code>	11
4.4	<code>CommandParser</code>	12
4.4.1	<code>split(inputString as String, Optional delimiter as String = ",", Optional maxItems as Integer = 255)</code>	12
4.4.2	<code>executeCommand(command as String)</code>	12
4.4.3	<code>commandWrapper(o as GwxPick)</code>	12
4.5	<code>ThisDisplay</code>	12
4.5.1	<code>GwxDisplay_PostRuntimeStart</code>	13
4.5.2	<code>BlinkenLights()</code>	13
4.5.3	<code>ArcnetScan()</code>	13
4.6	<code>Graphical Interface</code>	13
4.6.1	<code>GraphWorX32 in 60 Seconds</code>	13
4.6.2	<code>Window Layout</code>	14
4.6.3	<code>FEM/GCC Master Controls</code>	16
4.6.4	<code>FEM Channel Indicators/Controls</code>	16
4.6.5	<code>FEM Group Buttons</code>	16
4.6.6	<code>GCC Channel Buttons</code>	16
4.6.7	<code>What to Change</code>	21

5 Additional Resources **21**

6 About This Document **21**

List of Figures

1 Main control window 15

2 FEM/GCC On Button Property Inspector 17

3 FEM Button Color Configuration 18

4 FEM Button Pick Configuration 19

5 GCC On Button Pick 20

6 GCC On Button Hide 22

1 Introduction

The low-voltage control system software for muon tracking consists of two graphical user interface (GUI) windows, one each for the north and south arms, and a digital power-control setup located in the control racks. Since the system controls various types of equipment, which must be powered on in the correct order, the standard software controls contain logic to bring the system up and shut it down cleanly. If the system gets in an unclean state that the automatic controls cannot correct, or more granular control is required for diagnostics, a muon-tracking expert can override the master controls and enable or disable individual channels.

2 Operation

2.1 Quick Start

To bring the system up, open the appropriate window (NMTLV or SMTLV). Click the “Turn Cameras ON” button to activate the alignment cameras. Once the indicators have all changed to red, click the “Turn FEM/GCC ON” button. This will take a few seconds, as the FEMs are powered up and allowed to settle before the GCCs are stepped up. (If any GCCs are already on, they will be powered down before the FEMs are turned on.) When the blue processing light turns off, the system is ready.

To shut the system down, open the appropriate window (NMTLV or SMTLV). Click the “Turn FEM/GCC OFF” button. This will step the GCCs down in order and then power off the FEMs. Do not press any other buttons until the blue processing light turns off. After the FEMs are off, click the “Turn Cameras OFF” button. When all of the indicators have changed to green, the system is completely shut down.

2.2 Shift Personnel

The GUI, when first started, is in shift-operator mode. This mode provides access to the master controls for the front-end modules (FEMs) and G-link/C-link crates (GCCs) as well as the alignment cameras.

The Camera ON and Camera OFF buttons instantly activate and deactivate the entire camera subsystem as well as the signal converter for the cooling water-flow sensors on each rack, which were attached to the low-voltage distribution system with the cameras. No special operation is required for the cameras.

The FEM/GCC buttons cleanly activate and deactivate the FEM and GCC subsystem. Each FEM must come up before its associated GCC so that the GCC will detect it on powerup and start monitoring it; if the FEM is brought up after the GCC, it will be ignored. Therefore, if any GCCs are on when the powerup sequence starts, they are shut down first. Then the FEMs are brought up, the controller waits three seconds for them to initialize, and the GCCs are powered on. When shutting down, all GCCs are shut down and then all FEMs.

The GCCs themselves have both +5V and +3.3V channels that must be brought up in the correct order to avoid damage to the inductors on the cards. When turning on the GCCs, the controller first brings up the +5V channels, waits three seconds for them to stabilize, and then brings up the +3.3V channels. The inverse is performed on shutdown.

Any problem in startup or shutdown should be detected by the control scripts, which will ask what to do. In nearly all cases, retrying the step should allow the procedure to continue normally. *Any repeated error indicates a malfunction of the system and should be referred to an expert.*

Note that even in shift-operator mode, the controls for the individual FEM channels are active, since there is no way to disable the control without disabling the color indicators. *These controls should not be used by the shift operator.*

2.3 Muon-Tracking Experts

Checking the “Expert Mode” box enables, in addition to the master controls, controls for the FEMs and GCCs for each quadrant or octant and for each individual channel on the FEMs. Individual channel control has not been put in place on the GCCs because of the risk of damaging the hardware; the GCCs should always be brought up in the correct order, and the control logic is capable of dealing with error conditions that might arise.

Additionally, expert mode provides a set of controls for each slot on the ADAM module. Hovering a button will show the entire slot’s current state as a binary number, and clicking the button will turn the entire slot on or off (including channel 15 and any unused channels in 0–14). If using the slot buttons to recover from an error condition in the control system, it is usually best to power down the low-voltage crates in advance to prevent malfunction or damage to the hardware, since the slot buttons bypass the GCC safety logic.

3 Hardware Design

3.1 ADAM Network

The north and south control systems each have three ADAM-5000E system units that are mounted in their respective low-voltage racks. Commands and queries for the ADAM system go out of an RS-232C serial port on the controlling computer, through a fiber connection into the IR, and onto an RS-485 cable that is daisy-chained through several ADAM modules.

The modules are all entered in the OPC system slot-by-slot; the tags are in the form

NMT1-[ModuleNumber].Slot[SlotNumber]_DataWord

for north and

SMT2-[ModuleNumber].Slot[SlotNumber]_DataWord

for south, where ModuleNumber is from 4–6 and SlotNumber is 0-based. The last slot on the second low-voltage module in the south arm, therefore, is “SMT2-5.Slot7_DataWord”.

For performance reasons, logical access to the individual channel bits has been disabled; the only interface to the ADAM modules is through the 16-bit data word. The least significant bit is channel 0, at the top of the slot.

Note that the OPC server takes about 300ms to activate any command sent to the ADAM modules and that executing two commands in rapid succession may cause threading/concurrency problems.

3.2 ADAM Module Layout

Each of the ADAM system modules in the muon-tracking low-voltage control system has eight slots, numbered 0 to 7. All of these slots that are in use contain ADAM-5056D digital output modules with 16 open-collector output channels, numbered 0 to 15.

In this control system, only channels 0–14 on any given slot are used, since the low-voltage distribution (LVD) cards each have ten channels and the cleanest mapping between the ADAM system and the LVD is a 3:2 mapping, leaving the 16th channel unused.

3.3 Low-Voltage Distribution Cards

The distribution cards each have ten output channels and are controlled by a 14-pin block connector. Pins 1–10 are the control lines (normally open; short them to ground to turn them on), and pins 11–14 are shorted together as signal ground. Each card distributes a single voltage to all of its outputs, but the cards powering the FEMs and GCCs output +5V while those powering the alignment cameras output +12V. The cards each have a master power LED and LEDs for each channel.

4 Software Design

4.1 Overview

4.1.1 Introduction

The GraphWorX32 system is very limited in its built-in functionality and only supports scripting and extensions via Microsoft Visual Basic for Applications (VBA). In order to implement the specified behavior of having the FEMs turn on before the GCCs and having the 5V and 3.3V channels turn on and off in the correct order, we were required to write a modular control system in a language which does not support clean namespaces and other functionality. Much of the design of this system was imposed by limitations

in VBA; if you have questions or comments (or can think of ways to clean up the code), please contact the author.

The code is divided into two major libraries and a button-press handler. The libraries contain the general query/set utility functions, data structures for the controller, and a simple command parser written for the GCC on/off buttons. The contents of each module are detailed below.

4.1.2 General Notes

- Anywhere a data word is used, the VBA `Long` type should be used to hold it. The data word is a 16-bit value, but since VBA treats the 16-bit `Int` type as signed, problems will arise whenever type promotion occurs—your bits will get twiddled in a major way. Keep the word in a `Long` and this will not occur.
- Several of the functions employ `Goto` statements and labels. While the program's authors are aware that this is less than desirable, since VBA does not support exceptions (that we could find, at least), this was the cleanest way to implement the retry operation for error handling.
- Division of some of the functions into modules, especially between `GwxButtons_Main` and `GwxLibrary_Main`, is somewhat arbitrary. We tried to place generally reusable code in the latter and button-called code in the former, but this was complicated by the later need to add a command parser to the system. The function names should make it fairly clear what does what.
- The OPC server takes about 300ms to activate any command sent to the ADAM modules, and executing two commands in rapid succession may cause thread-concurrency problems.
- The FEMs must be brought up before the GCCs because the GCCs will not detect them otherwise, so having the FEMs off and the GCCs on simply means that the FEMs cannot be addressed. The 5V GCC channels, however, must be brought up before and shut down after the 3.3V channels to avoid damage to the GCC boards themselves. This means that the system will not wait as long for the GCCs to shut down as it will for the FEMs to initially come up.
- Visual Basic treats functions (which return values) and subroutines (which do not; think `void`) as separate groups—but not consistently; sometimes invoking a function requires treating it like a subroutine. The author is a C/C++/Java/Ruby programmer and calls them all functions. Beware the VBA `Call` statement.

4.2 *GwxLibrary_Main*

This module contains symbolic constants, the table of bitmasks for the channel groups, the `set` and `query` function groups, and a few odd utility functions.

4.2.1 Constants

The `GROUP` constants are a simple named enumeration for use in the `query_group` and `set_group` functions.

GROUP_ALL all active channels in the system (does not include spare or otherwise inactive channels)

GROUP_FEM all channels controlling FEM power

GROUP_GCC_3V channels controlling the +3.3V inputs to the GCCs

GROUP_GCC_5V channels controlling the +5V inputs to the GCCs

GROUP_GCC_ALL all GCC channels; that is, the union of the previous two sets

GROUP_CAMERA channels controlling the alignment cameras plus the ancillary water-flow converter box

The `QUERY` constants are a named enumeration for use in the `query_group` function specifying what type of query to execute.

QUERY_ANY return true if any of the specified channels is on; return false only if no channel is on

QUERY_ALL return true only if all of the specified channels are on; return false if any channel is off

The `WAITTIME` specifies how long, in seconds, the system should wait after bringing up the FEMs to bring up the 3.3V GCCs and then how long to wait to bring up the 5V GCCs, or how long to wait between bringing down the 5V and the 3.3V GCC channels. The system will wait at least this long and then will continue only if the command has been successful. The `TIMEOUT` specifies how long to watch the channels before deciding that they are not coming up (or down) and returning an error to the operator.

4.2.2 BITMASKS

The `BITMASKS` table specifies to the control program exactly which channels control what. The indexes to `BITMASKS` are module number, slot number, and group identifier (see 4.2.1), and the contents of each entry are a bitmask with the relevant channels set. For example, `BITMASKS(4, 0, GROUP_FEM)` is 32767 or 0x7fff, indicating that every channel on module 4, slot 0 controls a FEM (bit 15 is never used).

VBA does not provide a way to initialize variables analogous to C's `int i = 5;`, and so actually populating `BITMASKS` is done in the `initialize_data` function (see 4.2.3 on the next page).

4.2.3 initialize_data()

initialize_data is required to populate the data structures required by the system since VBA does not provide a way to initialize them at declaration. It is called automatically by

ThisDisplay.GwxDisplay_PostRuntimeStart (see 4.5.1 on page 13), which ensures that the setup is run every time the GUI window enters Runtime mode.

All of the actual data for BITMASKS are placed here, which is as non-hardcoded as can be practically done. *Any changes made to the channel mappings must be reflected here, both in any channels that may be added to a group and in channels that may be removed. Failure to update this table will cause interruptions in data acquisition and could result in hardware damage. See "What to Change" at 4.6.7 on page 21.*

4.2.4 query(module as Integer, slot as Integer)

The query function returns the 16-bit status word of the requested slot (see the note on data types at 4.1.2 on page 7). It may be used arbitrarily as needed but is primarily used for convenience by the query_group function, since reading a data word requires a cumbersome object acquisition.

4.2.5 query_group(group as Integer, query_mode as Integer)

The arguments to query_group should both be symbolic constants as defined at 4.2.1 on the page before. query_group iterates through all of the slots in all of the modules, checking the status of the requested group. Note that trying to read the data word of a nonexistent data-out module in an empty slot will result in a runtime error; this is avoided since query_group ignores any slot that does not have any "interesting" channels (a slot for which the BITMASKS entry is 0), which implicitly ignores empty slots. query_group returns true or false as described in the QUERY constant descriptions.

4.2.6 set_channels(module as Integer, slot as Integer, bitmask as Long, new_state as Boolean)

set_channels is a low-level function that sets or clears a group of bits. If new_state is true, the channels set in the bitmask will be turned on; otherwise, those channels will be turned off.

4.2.7 set_group(group as Integer, new_state as Boolean)

The set_group function is similar to set_channels, except that it operates on the higher level of channel groups. The first argument should be drawn from the GROUP constants listed at 4.2.1 on the preceding page. Note that like query_group, this function ignores slots without interesting channels.

4.2.8 gcc_startup()

This is the major-logic function that oversees bringing up the GCCs in the correct order; it returns true if the startup finishes cleanly and false if it is aborted. The individual commands are commented, so an overview of the process is provided here.

First, the `GCC_ALL` group is queried to see if all of the GCCs are already up. If so, the function exits early with a success code. Otherwise, if some of the 3.3V channels are already on (which should never happen), they are shut down.

Next, all 5V channels are brought up, but only if they are not all on already. The function waits at least `WAITTIME` but no more than `TIMEOUT` seconds for them all to come online (this should happen within about 300ms), and asks the operator what to do if some channels are stuck. Note that the operator is not given the option to bring up the 3.3V channels if the 5V powerup fails.

If no problem occurred, the 3.3V channels are brought up and the function exits with success. The function does not wait for the 3.3V channels to report having come up before returning (if there is a problem bringing up the 3.3V channels, the operator can retry the startup sequence without causing damage or loss).

4.2.9 gcc_shutdown()

This is the major-logic function that brings the GCCs down in the correct order; it returns true if the shutdown finishes cleanly *or if the operator chooses to ignore shutdown errors* and false if the operator chooses to cancel the shutdown. These return codes are used because the function cannot throw an exception on an error but still must report failures to its calling function. The individual commands are commented, so an overview of the process is provided here.

First, the `GCC_ALL` group is queried to see if any GCC is currently on. If not, the function exits early with a success code.

Next, all 3.3V channels are brought down, but only if at least one 3.3V channel is up. The function waits at least `WAITTIME` but no more than `TIMEOUT` seconds for them all to come down (this should happen within about 300ms), and asks the operator what to do if some channels are stuck.

If some 3.3V channels would not come down, the operator may choose to retry shutting them down, abort the entire shutdown, or ignore the error and crash the GCCs.

If, however, the 3.3V shutdown went cleanly, the function orders the 5V channels shut down, waits up to `TIMEOUT` seconds for the command to clear the queue on the OPC server (see the note on latency at 4.1.2 on page 7), and returns success.

4.2.10 set_led_color(r as Integer, g as Integer, b as Integer)

This function simply sets the color of the “Processing” light as an RGB value.

4.2.11 `wait(seconds as Integer)`

This function implements a polite wait function (yielding the processor instead of calling `sleep`, which in Windows apparently spins instead of sending the processor idle and blocks such things as actually sending the commands we're waiting for). Note that this function only has whole-second resolution, so calling `wait(1)` in something important is asking for trouble.

4.3 *GwxButtons_Main*

This module contains the functions called directly by the GUI buttons (except for the command parser).

4.3.1 `camera_startup` and `camera_shutdown`

These functions are simply wrappers to `set_group` and could very reasonably be included in the command parser.

4.3.2 `fem_gcc_startup`

This major-logic function supervises the entire startup sequence for the FEMs and GCCs. The individual commands are commented, so an overview of the process is provided here.

First, if everything is already on, the function exits early. Otherwise, if some GCCs are on but some FEMs are not, the GCCs are shut down. A problem in the GCC shutdown will abort the startup at this point.

Next, the FEMs are brought up if they are not all up already. The function will wait at least `WAITTIME` but no more than `TIMEOUT` seconds for the FEMs to come up. If some channels are stuck, the operator is given the choice to retry or to abort the shutdown.

Finally, `gcc_startup` is called. If the startup fails (the operator chooses to abort a failed startup), the FEMs are shut back down. Otherwise, the function exits with success.

4.3.3 `fem_gcc_shutdown`

This major-logic function supervises the entire shutdown sequence for the GCCs and FEMs. The individual commands are commented, so an overview of the process is provided here.

First, if everything is already off, the function exits early. Otherwise, the GCCs are shut down. If all of the GCCs are already off, this should return immediately.

If the GCCs actually must be brought down, there are two possible results: either the shutdown completes cleanly (or the operator ignores errors), or the operator aborts the shutdown. If the shutdown is aborted, `fem_gcc_shutdown` exits immediately.

Finally, the FEMs are turned off. The function exits without waiting for the command to clear.

4.4 **CommandParser**

This module was created to get around the obstacle that buttons (“Picks”) in GraphWorX32 can call functions but cannot pass parameters. They do have a space for “User Custom Data”, which can be read out by the function as a string. Since the GCC control buttons in particular have the need to execute complex command sequences, we wrote this command parser and placed the necessary commands in the User Custom Data field.

4.4.1 **split(inputString as String, Optional delimiter as String = “,”, Optional maxItems as Integer = 255)**

This function implements a basic string-split function, which for some reason is not included in VBA. It returns the `inputString` split at `delimiter` into an array of up to `maxItems` strings, discarding the delimiter.

Warning: While this function has been verified to operate correctly at low numbers of splits, it has not been tested for more than ten substrings. The `maxItems` parameter is provided because of a limitation in how VBA handles array allocation and may have bugs. If there is some reason that this function needs to be used for something more rigorous, more extensive testing should be performed.

4.4.2 **executeCommand(command as String)**

This is the master command executor of the module. It takes some sort of command as a parameter, splits it on the comma to get the command name and any parameters, and uses a `Select Case` statement to execute the desired command.

Currently, the only commands implemented are `GCCUP` and `GCCDOWN`.

Note that this command cannot be called directly from a GraphWorX32 Pick; use `commandWrapper`.

4.4.3 **commandWrapper(o as GwxPick)**

This function is solely a wrapper used to transform the Pick’s User Custom Data into a string suitable for `executeCommand`. To see this function in use, see the Picks on the per-octant GCC on/off buttons.

4.5 **ThisDisplay**

This module contains functions called implicitly by GraphWorX32 (if present) and a few testing support functions.

4.5.1 GwxDisplay_PostRuntimeStart

This function is called implicitly when the window enters Runtime mode. It is used here to initialize the BITMASK table and, during debugging, to call testing functions.

4.5.2 BlinkenLights ()

This function goes through every channel in ADAM-module order and flashes every camera or FEM channel. (GCC channels are not tested to prevent damage to the boards by bringing up the 3.3V without the 5V).

4.5.3 ArcnetScan ()

This function goes through every channel, much like `BlinkenLights`, except that it brings up the controlling GCC (so that the ARCnet nodes are powered) and holds every channel online until released. This is to facilitate checking the ARCnet system by bringing up one channel, scanning the ARCnet network, recording which FEM(s) or cameras responded, and then going on to the next channel.

4.6 Graphical Interface

4.6.1 GraphWorX32 in 60 Seconds

GraphWorX32 is a glorified UI builder that provides connections to OPC data sources/sinks. The names of some of the components seem to have been poorly translated, so here are the few most important:

Button implicitly applied to some objects, such as rectangles, when Picks are added to them

Color applied to a filled shape, changes the color based on some data

Hide allows hiding or disabling the object (used for Expert Mode)

Pick “action”; makes clicking on the object do something

The Pick provides the option to directly manipulate a data source (used for the FEM control buttons and the expert slot buttons) or call a function (see the Command Parser at 4.4 on the preceding page).

GraphWorX32 is somewhat scriptable using VBA (press Alt-F11 for the editor) and has hooks for acting on programmatic events (see 4.5 on the page before).

Nearly everything we have set up is in the GraphWorX32 help system, but the documentation is rather opaque at times. If you can't figure out why we did something, ask us or simply voodoo program it (see The Jargon File at <http://catb.org/~esr/jargon/html/V/voodoo-programming.html>).

4.6.2 Window Layout

The main control windows for north and south are nearly identical; all differences reflect the physical setup of the arms. For the purposes of this documentation, we will treat only the north arm.

The main window is divided into several sections; the primary ones are the FEM indicators/buttons, the GCC indicators/buttons, the master FEM/GCC buttons, and the camera indicators/buttons. The entire window is shown in figure 1.

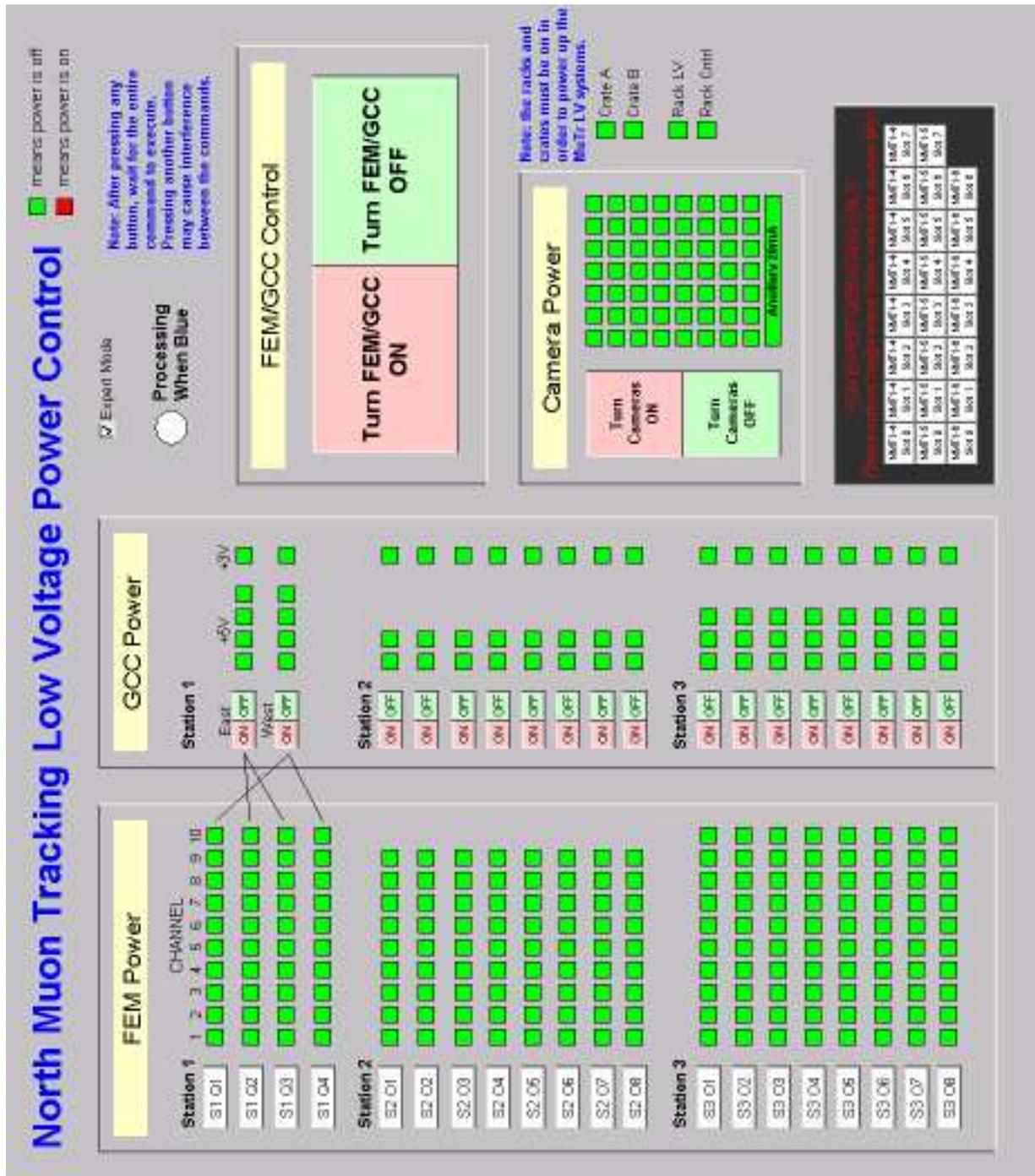


Figure 1: Main control window

Many of the indicators and buttons are grouped together into one logical unit (such as all the cameras or all the FEMs in one octant) to make window placement more uniform. Use Ctrl-U to ungroup objects if

they need editing and Ctrl-G to regroup them when finished. If they are accidentally knocked out of place, Ctrl-Z (undo) will put them back.

The Expert Mode checkbox controls whether the FEM and GCC octant buttons are active and whether the slot buttons are visible. Checking it sets the GraphWorX32 local boolean variable `~~expert~~` and unchecking the box clears it.

The slot buttons should only be used for diagnostics and should only be clicked when the power to the crates is off, as they immediately turn on all channels in a slot, which could cause damage to a GCC.

The crate and rack indicator lights are provided as a convenience and may be removed or disabled if required without interfering with the rest of the window.

4.6.3 FEM/GCC Master Controls

The two large control buttons for the FEM/GCC subsystem simply call the `fem_gcc_startup` function (see 4.3.2 on page 11). The Property Inspector dialog for the on button is shown in figure 2 on the next page.

4.6.4 FEM Channel Indicators/Controls

Each light in the FEM section represents a single channel. The colors are determined directly from the contents of the appropriate data word by the use of a boolean AND operation. Changing any of these buttons requires changing the bitmask in the Color and in both Pick fields (as well as the other references; see “What to Change” at 4.6.7 on page 21). In the sample Color configuration given in figure 3 on page 18, it can be seen that the channel in question is on channel 10 ($2^{10}=1024$) of module 4, slot 2. The Pick for the same button (figure 4 on page 19) shows both the syntax required to write a calculated value instead of an immediate value on clicking (required since this channel’s bit must be changed while not disturbing the other channels in this word) as well as the format of the boolean expression (C-style).

4.6.5 FEM Group Buttons

These buttons (only available in Expert Mode) turn on or off all channels for a single set of FEMs. They use one or two simple Picks with bitmasks specifying all channels to be changed.

4.6.6 GCC Channel Buttons

These buttons (only available in Expert Mode) will cleanly bring up or shut down the power for a single GCC unit or for the GCCs for an entire station. They call the `GCCUP` and `GCCDOWN` command in the Command Parser (see 4.4 on page 12). In the example shown in figure 5 on page 20, the GCC connected to module 5, slot 6, using channels 0–2 is brought up. The `GCCUP` command can determine which of the channels are 5V channels and bring them up before bringing up the 3.3V channels. (`GCCUP` will ignore any non-GCC channels, but it will happily proceed to power up *all* 5V and *all* 3.3V channels specified in the

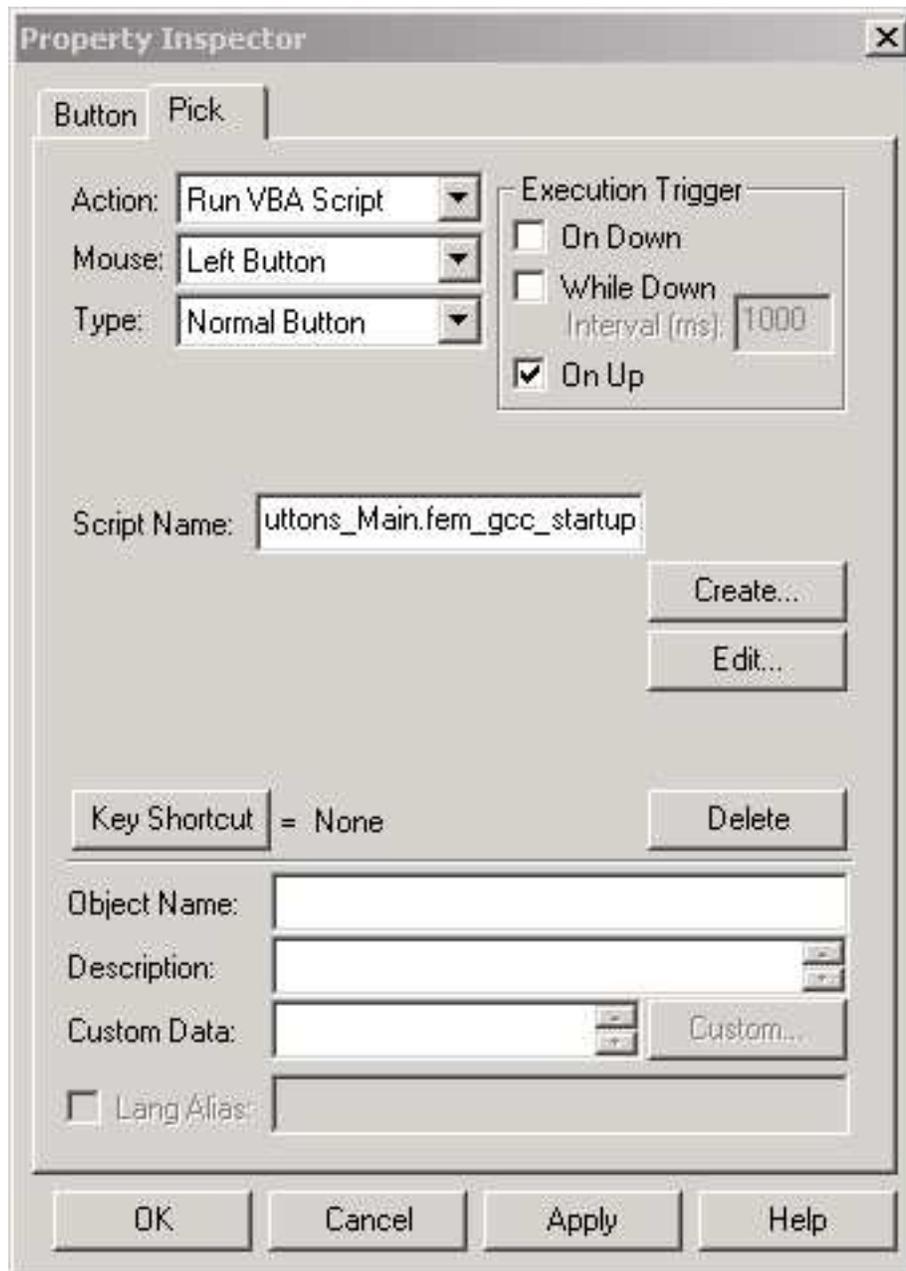


Figure 2: FEM/GCC On Button Property Inspector

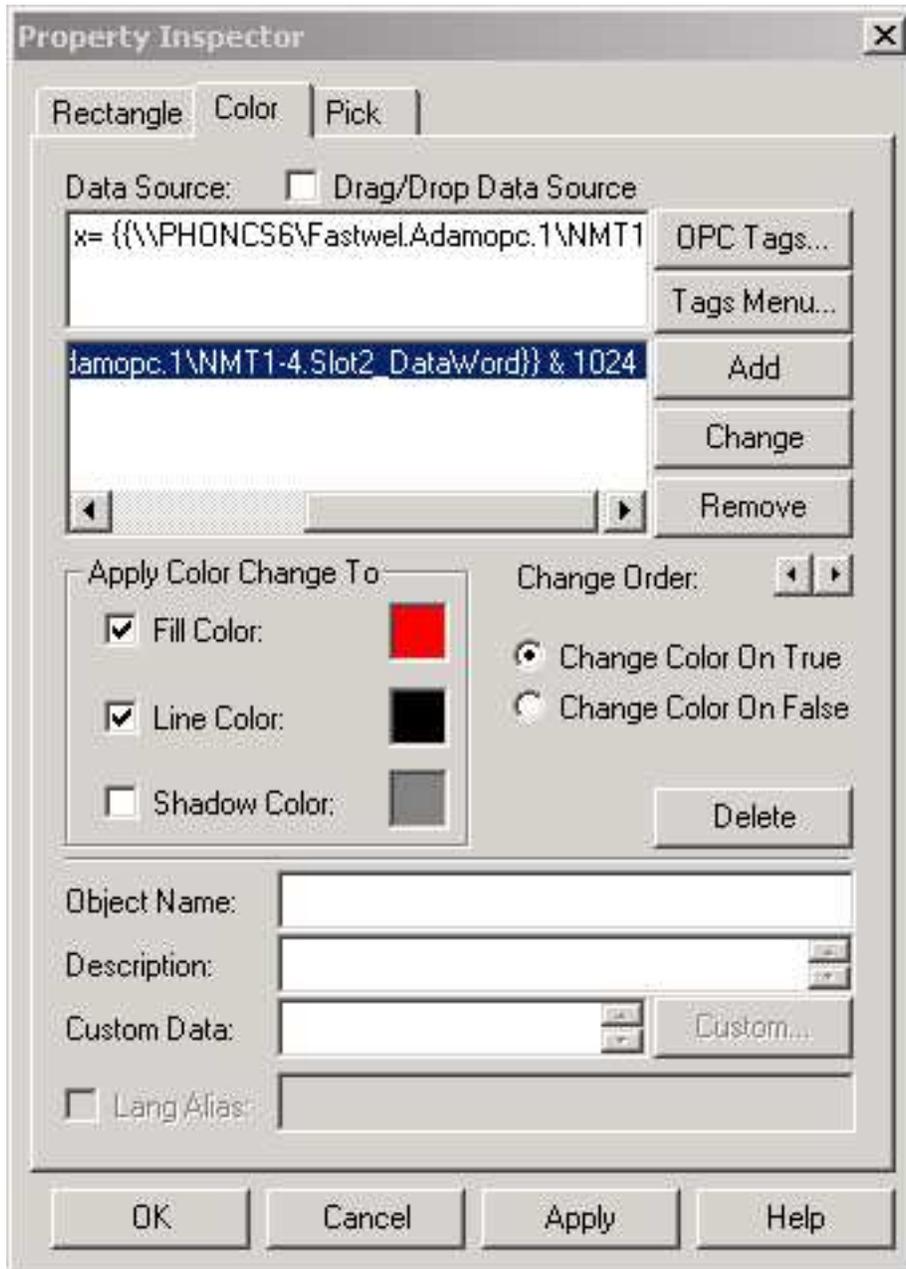


Figure 3: FEM Button Color Configuration

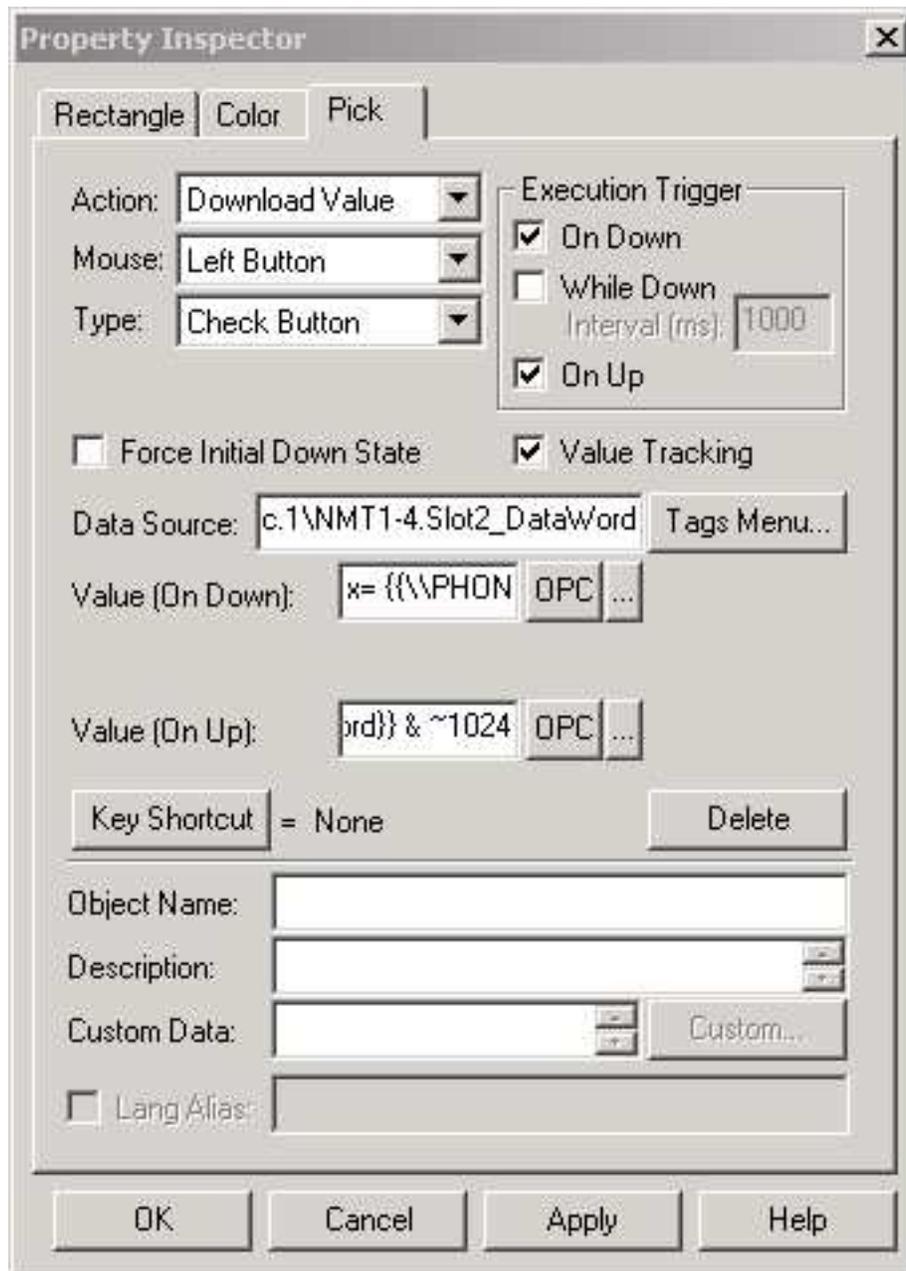


Figure 4: FEM Button Pick Configuration

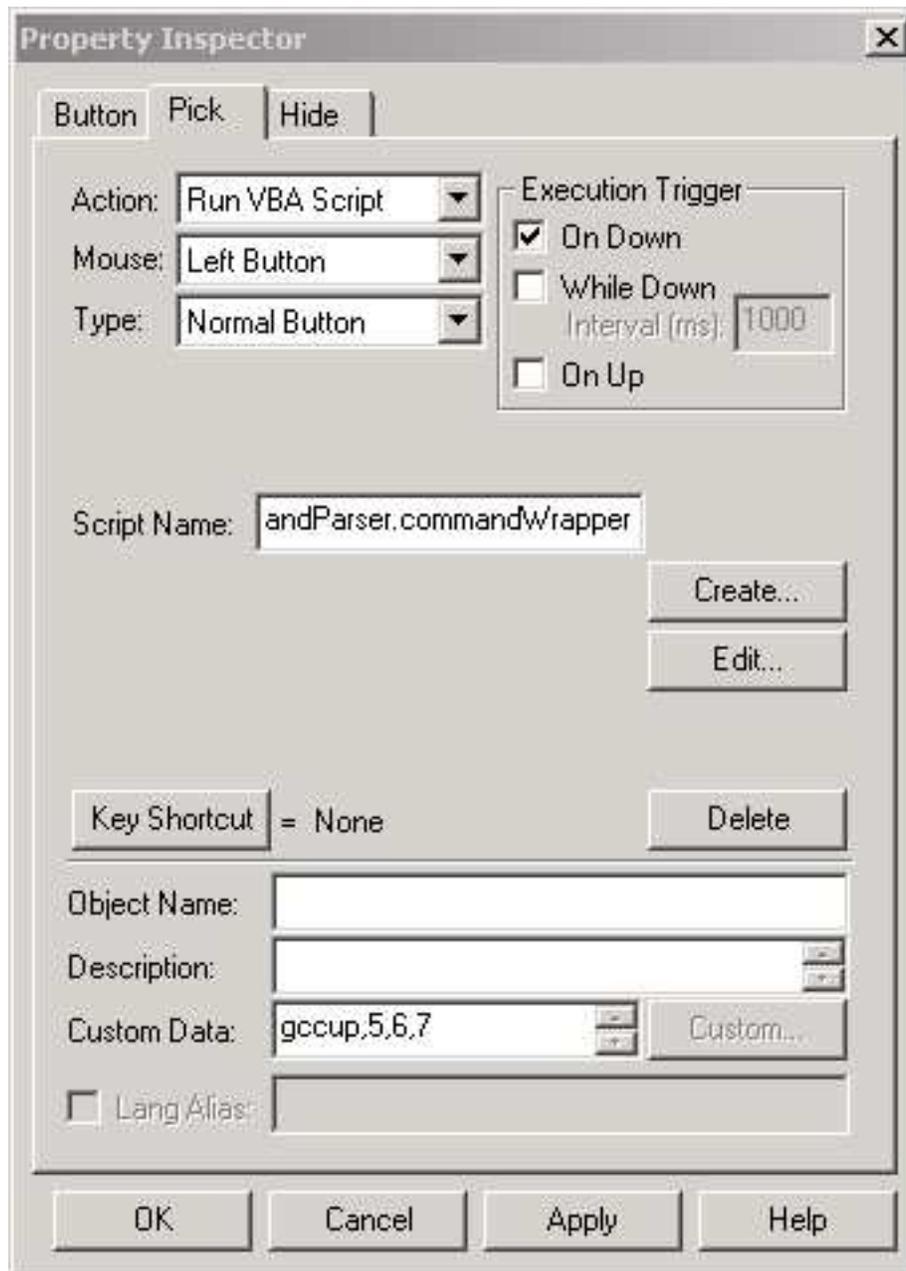


Figure 5: GCC On Button Pick

parameters; ensuring that the bitmask actually specifies the channels for the GCC or GCCs in question is the responsibility of the caller.)

For completeness, figure 6 on the next page shows the setting used to disable the buttons when not in Expert Mode.

4.6.7 What to Change

Since GraphWorX32 does not provide for a convenient central repository for arbitrary user data, many of the parameters used by this control system are specified in multiple places. The following list shows all the places where changes must be made to keep the system operating correctly.

FEM Channels FEM indicator button (Color and both Picks), FEM quadrant/octant button, BITMASKS

GCC Channels GCC indicator button (Color only), GCC On and Off buttons (parameters to GCCUP and GCCDOWN commands in the User Custom Data), BITMASKS

Camera Channels camera indicator light (Color only), BITMASKS

Crate/Rack Monitoring appropriate indicator light (Color only) [indicator light is provided as a convenience to the operator and is not required for the control system to operate correctly]

5 Additional Resources

For the latest maintenance information, see the Muon Tracking Logbook at http://www.phenix.bnl.gov/phenix/WWW/publish/leitch/public_html/.

The charts for the cable mappings can be found at https://www.phenix.bnl.gov/WWW/p/draft/towell/MuTR_LV/.

If these documents do not answer your question, feel free to e-mail the authors at bnl@christophersmith.net or mbd02a@acu.edu.

6 About This Document

This document was created using LyX (a very convenient semi-graphical L^AT_EX editor) and exported to L^AT_EX and then to various other formats. It would be preferable to make any necessary changes to the LyX file itself. If that is not possible, then changes should be made to the L^AT_EX source file *and not to the resultant HTML or other documents*.

The command used to produce the HTML output was

```
latex2html -split 4 -noinfo PowerControl.tex
```

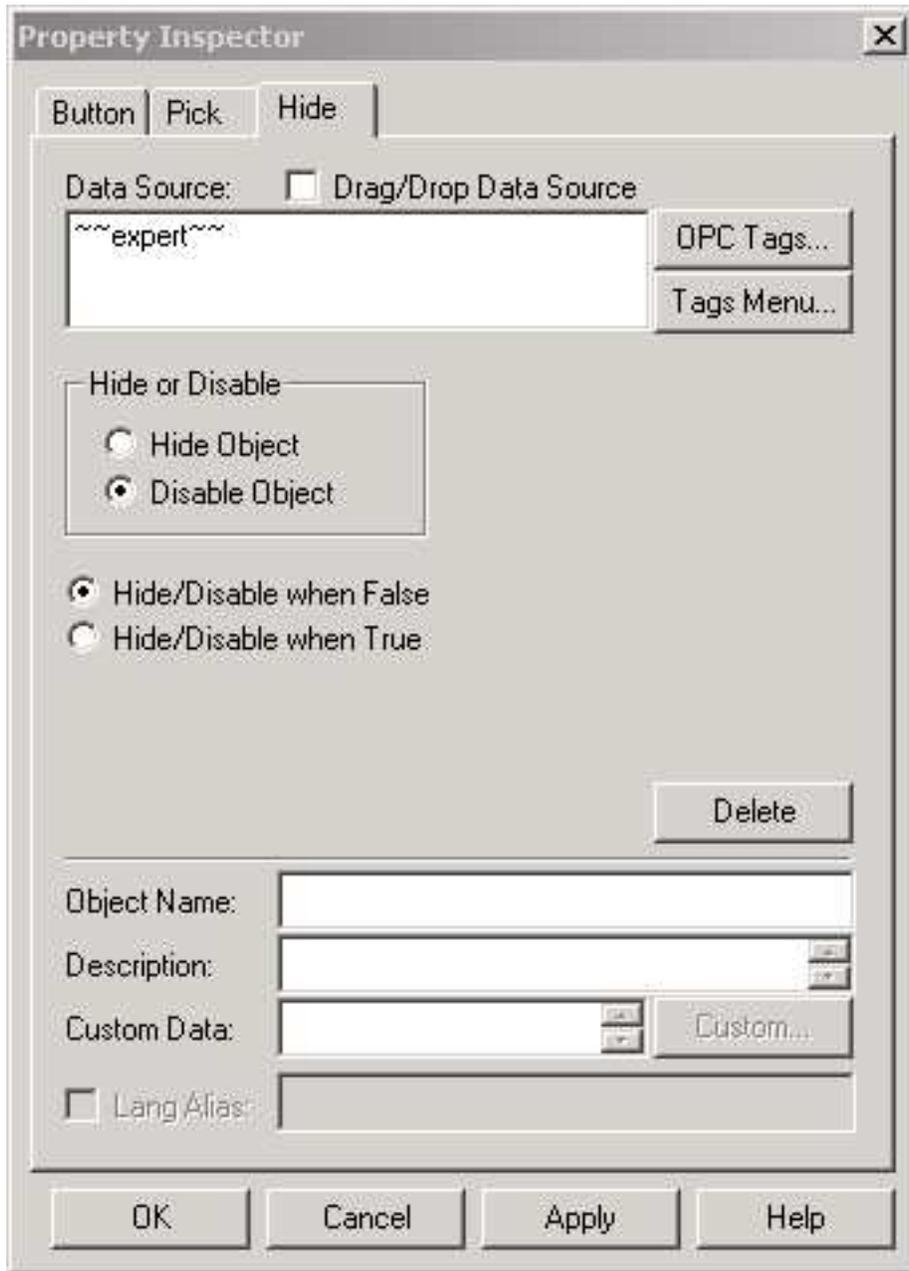


Figure 6: GCC On Button Hide

Apparently `latex2html` has problems with cross-references; any tips on how to make it render them correctly would be appreciated.