MICAS-X Programming Manual

version 2.1

Table of Contents

1 Introduction

The MICAS-X system consists of numerous components. The MICAS-X.vi VI is the main control program. It starts all the other components, including Instruments, Modules, Drivers, and Utilities. The MICAS-X.vi VI has 18 tabs and a header area above the tabs. General controls and indicators that need to be seen and accessed for all tabs are in the header area. The tabs include the Control tab, 14 Display tabs, the Utility tab, the MICAS-X tab, and the Debug tab. Note that many of these tabs can be configured to have different names than these default names.

- The Control tab is a main control and summary tab. It includes Controllers, Switches, Sequences, and Triggers, as well as a channel list and an optional time-series graph.

- The Display tabs are only shown as needed. Instruments and Displays can be loaded in these.

- The Utility tab is where the operator can access various utility programs as needed. These are loaded on request, rather than when the program starts. Utilities are used for configuration of MICAS-X, data review, log file review, and other similar tasks.

- The MICAS-X tab is a status page. It displays information on the state of the MICAS-X program, including the most recent errors and log entries, configuration and data file names, a list of channel values, etc.

- The Debug tab is normally closed. It is intended for debugging the program during development.

Most of MICAS-X is designed to run within the main MICAS-X VI. Due to the dynamic, configurable nature of MICAS-X, the various front panels used throughout the program are very carefully laid out. Many controls are hidden and only made visible when the configuration requires them. Because of this, the MICAS-X front panel is set to not allow maximizing or resizing, and scroll bars are turned off. It is HIGHLY recommended that these settings not be changed. Although this design makes it possible for MICAS-X to adapt to a wide range of configuration, the down side is that it limits the operator's ability to lay out displays in completely arbitrary ways. All Instruments, Displays, Utilities, and Configuration Editors must be sized exactly right in order for the to work within MICAS-X.

One partial exemption to this limitation is that Instruments and Displays can be configured to open their front panels Outside of the main MICAS-X VI. This allows the operator to have multiple windows open at once,to position them around the screen as desired, and make them any size desired. Note that Instruments and Displays that are designed this way must still adhere to the MICAS-X definitions in all other respects in order to work with the MICAS-X system. Also note that configuring an Instrument or Display to open Outside of MICAS-X has a potential performance effect. When an Instrument or Display is opened inside of MICAS-X, MICAS-X includes logic that ensures that the Instrument or Display front panel is only updated when its tab is the current tab. This greatly reduces CPU loading by preventing graphs and other displays that are not being viewed from being unnecessarily updated. When an Instrument or Display is opened Outside of MICAS-X, this feature is disabled, and the Instrument or Display front panel is continuously updated. This may or may not impact the program's performance, depending on what the Instrument or Display is presenting and how it is programmed.

2 Configuring the Font Size in LabVIEW

MICAS-X was initially developed on Windows XP in LabVIEW 2012. Its development soon moved to Windows 7, and it is now written in LabVIEW 2013. Windows 7 typically uses a different set of font sizes and screen settings than XP, with the result that LabVIEW programs written in XP end up distorted in Windows 7, so that text labels are larger than intended, run into other items on the window, etc.

Several steps have been and can be taken to resolve this issue. When MICAS-X is installed as an executable, it installs with a .ini file that adjusts the fonts appropriately.

When MICAS-X is used within LabVIEW, however, the LabVIEW environment must be adjusted. It is best to do this before opening MICAS-X. If MICAS-X is opened before this adjustment is made, and any VIs are saved, the text on the VI front panels will be distorted. If this happens, it is best to adjust the settings, then download a fresh copy of MICAS-X.

To adjust the font settings in LabVIEW, select the Options item from the Tools menu. Click on Environment in the left pane of the resulting dialog, then in the right pane scroll down to Fonts. Uncheck the “Use Default Fonts” checkbox. With “Application Font” selected in the left item, click on Font Style. In the resulting pop-up window, change the Size to 13, then click OK. Repeat this process for the Dialog Font and System Font options in the left item. When all three fonts are set to a size of 13, click on the OK button at the bottom of the window.

3 General Guidelines

When developing LabVIEW code for the MICAS-X system, the following guidelines should be adhered to.

- All VIs will have a description adhering to the format in the templates.
- All VIs will have an icon based on one of the template icons.
- Every control and indicator that an end-user can see will have help/description text.
- The naming conventions for VIs and paths described in sections below must be adhered to in order for MICAS-X to work.
- General LabVIEW best practices will be followed, including those above for documentation, as well as the proper use of sub-VI's, limited diagram size, etc.

4 Debug Mode

When the MICAS-X program is running, it is possible to access a debug mode that allows more control over the program itself. To enable debug mode, click on the Summary tab, then click on the small black square in the upper right corner. A Debug tab will appear to the right of all other tabs. There is a data channel that is logged named “Debug” that is a 0 when MICAS-X is in normal mode, and 1 when it is in Debug mode.

Several features are available in Debug mode. When the Configuration Editor is run inside MICAS-X as a Utility and MICAS-X is in Debug mode, each editor page has an additional button. To the right of the Reset to Defaults button will be found a Save as Defaults button. Pressing this button will save the current settings to a special Defaults file. Whenever a “Reset to Defaults” button is pressed in any editor, the program first looks to see if such a Defaults file exists for the editor being used. If it does, the defaults are loaded from the file. If not, it loads defaults that were programmed into MICAS-X directly. In this way, default values for various components can be created for each installation of MICAS-X, allowing end users to revert to defaults that are meaningful for their system.

The other major feature of Debug mode is the Debug tab. This page allows the user to observe some of the tracking information inside MICAS-X and to access additional debugging resources. Some of these resources are useful only in the source code version of MICAS-X. For instance, opening a diagram of the main VI is not possible in the executable version, but in the source code version, diagrams can be opened and probed during execution. In particular, be aware of the "Show Diagram" button in the Program Queue tab. This will open the diagram of MICAS.vi, so that you can see how any feature of MICAS-X was programmed. This diagram is password protected, so it cannot be opened normally in LabVIEW's edit mode unless you have the password. When opened in debug mode, you still cannot edit the diagram, but you can view any part of it to learn how to accomplish many complex tasks with the MICAS-X system.

5 Directions for Creating a New Driver

- Decide on a name for the new Driver. In this example, the new Driver is named NIDaqDO.

- Find a similar Parameters cluster type def and use Save As to save a copy in a subfolder of the Drivers folder. Adhere to the naming convention. This cluster for this example is named NIDaqDO Parameters.ctl and was copied from the NIDaqDA Driver.

- Edit the icon to change the Driver name there-in.

- Edit the VI Properties/Description to change the name and make any other required changes.

- Use the body of the description of the VI as the description for the cluster by right clicking on the cluster.

- Update the label of the cluster.

- Edit the cluster contents as needed to create the configuration parameters for the new Driver. Note that Channels, Controllers, Sequences, Triggers, Commands, and Lists are always stored as strings, not rings, in the parameters and configuration files.

- Be sure to edit the Description for all the controls and indicators in the cluster, as well as on all MICAS-X VI front panels. (Note that for arrays, only the array container needs a valid description. In general, the contents of the array can inherit the description of the array container.)

- Open the XXXX Read Parameters.vi for the Driver that is serving as the template and use Save As to save a copy in the new Driver folder. In this case it is saved as NIDaqDO Read Parameters.vi. Edit the icon and description appropriately. Replace the parameters cluster on the front panel with the one for the new Driver. On the block diagram, replace the parameters cluster constant with the correct one for the new driver. Also, edit the Driver name in the string constant in the lower center of the diagram.

- Repeat the above steps for the XXXX Write Parameters.vi, except that there is no cluster constant on the diagram to replace. Do make sure to edit the Driver name in the string constant on the diagram.

- Open the MICAS-X Editor Template.vi. Use Save As to save it with the new Driver name in the Editors directory. In this case, it was named NIDaqDO Editor.vi. If it is not named correctly, the MICAS-X program will be unable to use it. Use the Add to Project option and move this item to the Editors folder in the project. Open the MICAS-X VI Tree.vi and add the editor to the diagram in the row of Editors. Edit the icon and description.

- Open the editor diagram and go to the Initialize case. Replace the parameters cluster constant with the one for the new Driver. Review the values of the constants in this cluster, as they will be used as defaults if the config file being read does not have the section, and if the default config file does not exist.

- Go to the Read File case and replace the Read File VI with the proper one. Right click on the grey "Program Parameters" in the bundle node and select the parameters cluster for this driver. Add another unbundle line to the unbundle node and select Prefix as the item. Wire this into the top of the Read Parameters VI.

- Go to the Write File case. Right click on the "Program Parameters" in black in the unbundle node and select the parameters for this driver. Then replace the Write Parameters VI with the proper one. You may need to delete and rewire the wire between those two nodes if it remains broken. Add another unbundle line to the unbundle node and select Prefix as the item. Wire this into the top of the Write Parameters VI. Make sure to retain the MICAS-X Write Config TimeStamp.vi in the Write File case, as all Editors must call this VI.

- Go to the Display case and add code to display the relevant parameters as they are read from the configuration cluster stored on the shift register. Channels related to other Drivers and stored as strings will need to be converted to rings, but Channels being defined and named for this Driver must be strings. See other drivers for examples.

- Go to the Wait case and add events that put new values of the parameters into the cluster on the shift register. Also add events for other functions, such as Delete, Insert, etc. See examples. Events that alter the contents themselves (such as Delete or Insert) must call the Display case via the queue. Events that respond to user changes to the parameters need not call the Display case. Both types of events must call the "Enable S&R" case.

- Create the "Names" VI. Open the "XXXX Names.vi" for a similar Driver and use the Save As feature to save it to the new Driver's directory with the appropriate name. Use the "Add to Project" option to add this VI to the project, then move it into the Drivers folder. The naming convention must be strictly adhered to. Name this VI "XXXX Names.vi" where "XXXX" is the name of the new Driver. Edit the icon and description appropriately. Open the diagram and edit the Driver Name in the string in the upper left. Replace the Read Parameters VI with that for the new Driver. Parse the configuration parameters appropriately to get a list of Channels (all channels) and Controllers (output channels, if there are any). Add this VI to the VI Tree.

- The Editable Names array must be filled with the Channel Names or roots of the Channel Names that can be edited by the end user. The Prefix must not be included in these editable names. For some Drivers, every channel name can be defined in the configuration editor. These would all be editable names. (See the NIDaqAD Driver as an example.) For other Drivers, there is a root name that the user can edit, and which the Driver then adds modifiers to to arrive at the Channel Names. In this case, only the root name would be put into the Editable Names array. (See the Array Driver as an example.) For yet other Drivers, the names are all hard-coded in the Names VI. For these cases, the Editable Names array should be empty.

- The Names VI also supplies MICAS-X with the list of custom Commands that the Driver will support (if any). Commands cannot have spaces in them and must be static; e.g. Command names cannot be created depending on the configuration options. Finally,, note that Commands must have the Prefix prepended to them in the Names VI, just as the Channels and Controllers do.

- Finally, the Names VI must contain the Info array on its front panel, which is also wired to the connector pane. This array is reserved for future use. It allows Drivers to supply information to other parts of MICAS-X, so that features of Drivers that have not yet been imagined or implemented can be supported without having to alter every Driver in MICAS-X. See the Info section later in this chapter for information on how this set of parameters is to be used.

- After completing these steps, you should have a working configuration editor for the new Driver. However, this Driver will not show up in the list of Drivers in the configuration editor until the Driver itself has been created.

- Now to create the actual Driver. Open a related Driver to use as a template and use the Save As feature to create a new VI with the new Driver's name, located in the Driver subfolder of the same name. Use the extension ".vi" if the Driver can only be instantiated once. Use the extension ".vit" if the Driver can be instantiated more than once within a single MICAS-X configuration. Use the "Add to Project" option to add it to the MICAS-X project, and then move it to the Drivers folder in the project. Edit the icon to change the Driver name. Edit the description appropriately.

- Note that ".vit" makes the VI a VI Template. When this is called by LabVIEW, a new copy is automatically created. This mechanism makes it possible for LabVIEW to call the same VI from the disk multiple times and end up with multiple independent copies in memory. This is useful for Drivers such as NIDaqAD, since a separate instance of the Driver can be used for each NI Daq Device connected to the computer. However, this must be avoided in certain cases in which the Driver is using a specific non-sharable resource. For example, if the driver accesses a global variable or a specific file, it may not work as a .vit since all the copies of the Driver would be accessing the same global or file.

- Edit the block diagram of the driver. Note that the Driver State control cannot be altered. All the functions of the Driver must fit within the framework provided by the commands defined by this State type-def. For most Drivers the Acquire (get data from the hardware and make it available to MICAS-X) and Set (set the value of an output channel) will suffice. Input-only devices (such as NIDaqAD) will not even need to use the Set command. If some Driver functionality does not fit within those commands, the custom "Command" command must be used. This allows MICAS-X Commands (from Sequences, Switches, or Triggers, but not yet from CCL) to activate specific custom Commands for Drivers. The Timers Driver includes custom commands and can be used as a reference, but in other respects it is an unusual Driver and should not be used as a direct model for other Drivers. Custom Driver Commands should not contain spaces. Whenever Custom Driver Commands are added, the VI "MICAS-X Select Config Display Options.vi" in the directory C:\MICAS-X\Configuration should be updated. This VI is used by configuration editors to determine which parameters should be displayed for each Command.

Each Driver gets a reference to its own copy of the MICAS-X Driver Data.vit. This is another VI Template, such that each time it is opened it creates a new copy, hence each Driver has a place to keep its most recent data. The Acquisition loop only has to query these .vit's to get the most recent Driver data. This allows Drivers to be run in numerous independent loops at different rates and timings, with the Acquisition loop always getting each Driver's most recent data. It is for this reason that each Driver also gets it's own Time channel prepended to the list of channels. (Note that the Time is prepended to the data array in the Acquire and Set cases, but is not explicitly included in the list of Channel names maintained inside the Driver.)

For Drivers with output channels (Controllers), the Set case is used to set a channel. An array on the shift register maintains the values of all the Controller channels, so that one Controller channel can be changed in memory and then all the Controller channel values can be sent to the hardware. This list of current Controller channels is also maintained in a VI called MICAS-X Controllers.vi. This VI knows all the most current values of all Controllers in MICAS-X. This is so that Controller values on the user interface can be reliably updated without race conditions. If the user interface was updated from the main Channels list of values, race conditions could occur in which a user changed a controller value, and then the user interface replaced the new value with the old value in the Channels memory before the Channels memory was updated with the latest value.

For Drivers with Commands, the Command case handles the execution of commands. See the Timers Driver for an example. Note that the Command selector case structure within the Commands case should be configured to recognize the Commands formatted in all CAPS.

When new Driver Commands are added, additional steps are required to optimize the user interface of the configuration editor for that Driver. If the command is not registered properly with MICAS-X, then the configuration editor will not know which parameters that Command requires. Hence, instead of a Channel list for the Command string parameter, it will only present a blank string, and the user will need to carefully enter the Channel name, spelled exactly right. However, by registering the Command with MICAS-X (in the MICAS Select Config Display Options.vi), the configuration editor can present a drop-down list of Channels when the Command needs a Channel as a parameter. The MICAS Select Config Display Options.vi is only to be edited by Original Code Consulting, however, so please contact OCC if you wish to register a new Command for MICAS-X.

6 Info Parameters

As mentioned above, the Driver Names VI must include an Info array as an output wired to its connector pane. This indicator is an array of clusters. Each cluster contains a string “Name” and a variant “Info Data”. The Name parameter defines what type of extra information the Driver can supply. The specific information being supplied is contained in the Info Data variant.

The purpose of the Info parameters is to allow the MICAS-X Driver model to be extensible, so that future Drivers can include new functionality without having to rewrite all existing Drivers to accommodate that functionality. The first example of this is the Controllers Driver. This Driver sends a “Mode” cluster out the Info parameters. The Mode cluster tells MICAS-X and various Displays that some channels may have a Mode Channel associated with them. By searching all the Drivers included in a configuration for which have this Mode Info parameter, MICAS-X can find all channels that have associated Mode channels and can accommodate them.

For the Mode parameter, the Controllers Driver creates an array of clusters. Each cluster contains two items, the name of a channel, and the initial value of that channel's associated Mode channel. Using this information MICAS-X can display the Mode channel controls next to the relevant channels.

A future use for the Info parameters may be to allow Drivers to define histogram channels. A Histogram Info parameter could include the first data channel in the histogram, how many total channels contain the histogram data, and information on the x-scaling of the histogram bins. This would allow a Display to create histogram displays from the data from any Driver that supported this Histogram Info.

7 Directions for Creating a New Instrument

- Decide on a name for the new Instrument. In this example, the new Instrument is named SPTOFMS. The Instrument should have a sub-folder of the Instruments folder with the same name as the Instrument itself. E.g. this example Instrument will be stored in the directory C:\MICAS\Instruments\SPTOFMS.

- Find a similar Parameters cluster type def and use Save As to save a copy in a subfolder of the Instrument folder. Adhere to the naming convention. This cluster for this example is named SPTOFMS Parameters.ctl.

- Edit the icon to change the Instrument name there-in.

- Edit the VI Properties/Description to change the name and make any other required changes.

- Use the body of the description of the VI as the description for the cluster by right clicking on the cluster.

- Update the label of the cluster.

- Edit the cluster contents as needed to create the configuration parameters for the new Instrument. Note that Channels, Controllers, Sequences, Triggers, Commands, and Lists are always stored as strings, not rings, in the parameters and configuration files.

- Be sure to edit the Description for all the controls and indicators in the cluster, as well as on all MICAS-X VI front panels. (Note that for arrays, only the array container needs a valid description. In general, the contents of the array can inherit the description of the array container.)

- Open the XXXX Read Parameters.vi for the Instrument that is serving as the template and use Save As to save a copy in the new Instrument folder. In this case it is saved as SPTOFMS Read Parameters.vi. Edit the icon and description appropriately. Replace the parameters cluster on the front panel with the one for the new Instrument. On the block diagram, replace the parameters cluster constant with the correct one for the new driver. Also, edit the Instrument name in the string constant in the lower center of the diagram.

- Repeat the above steps for the XXXX Write Parameters.vi, except that there is no cluster constant on the diagram to replace. Do make sure to edit the Instrument name in the string constant on the diagram.

- Open the MICAS-X Editor Template.vi. Use Save As to save it with the new Instrument name in the Editors directory. In this case, it was named SPTOFMS Editor.vi. If it is not named correctly, the MICAS-X program will be unable to use it. Use the Add to Project option and move this item to the Editors folder in the project. Open the MICAS-X VI Tree.vi and add the editor to the diagram in the row of Editors. Edit the icon and description.

- Open the editor diagram and go to the Initialize case. Replace the parameters cluster constant with the one for the new Instrument.

- Go to the Read File case and replace the Read File VI with the proper one. Right click on the grey "Program Parameters" in the bundle node and select the parameters cluster for this driver. Add another unbundle line to the unbundle node and select Prefix as the item. Wire this into the top of the Read Parameters VI.

- Go to the Write File case. Right click on the "Program Parameters" in black in the unbundle node and select the parameters for this Instrument. Then replace the Write Parameters VI with the proper one. You may need to delete and rewire the wire between those two nodes if it remains broken. Add another unbundle line to the unbundle node and select Prefix as the item. Wire this into the top of the

Write Parameters VI. Make sure to retain the MICAS-X Write Config TimeStamp.vi in the Write File case, as all Editors must call this VI.

- Go to the Display case and add code to display the relevant parameters as they are read from the configuration cluster stored on the shift register. Channels related to Drivers and stored as strings will need to be converted to rings.

- Go to the Wait case and add events that put new values of the parameters into the cluster on the shift register. Also add events for other functions, such as Delete, Insert, etc. See examples. Events that alter the contents themselves (such as Delete or Insert) must call the Display case via the queue. Events that respond to user changes to the parameters need not call the Display case. Both types of events must call the "Enable S&R" case.

- Now to create the actual Instrument. Open a related Instrument or the Instrument Template to use as a template and use the Save As feature to create a new VI with the new Instrument's name, located in the Instrument subfolder of the same name. Use the extension ".vi" if the Instrument can only be instantiated once. Use the extension ".vit" if the Instrument can be instantiated more than once within a single MICAS-X configuration. Use the "Add to Project" option to add it to the MICAS-X project, and then move it to the Instruments folder in the project. Edit the icon to change the Instrument name. Edit the description appropriately.

- In the Initialize case, change the log entry about starting the Instrument to contain the Instrument name. Do the same in the Stop case.

- In the Initialize case, change the name of the Display Queue to the Instrument name.

- In the Initialize case, add the Read Instrument Parameters VI, or replace the existing VI with the one for this Instrument. Also add or replace the Parameters constant with the one for this instrument.

Note that there is a Display Queue for every Instrument and Display, which MICAS-X uses to turn the display function on and off and to stop the module. Since Instruments and Displays can be multiply-instantiated, this queue must have a name that includes the Prefix for the Instrument or Display. To get the queue name, the prefix is added before the Instrument or Display name, with no space between them, and with no extension (.vi or .vit) on the Instrument or Display name.

There are VI's in the FlexMotion libraries that are contained in a .llb and which have illegal file name characters in them. This is fine when they are used within the LabVIEW system, but they cannot be added to a build. To be able to build the NIMotion Driver, two of these VI's must be renamed within the LabVIEW environment. They are

C:\Program Files (x86)\National Instruments\LabVIEW 2012\vi.lib\Motion\FlexMotion\FunctionsVIs\AxisResourceConfig.llb\Load Counts/Steps per Revolution.flx and C:\Program Files (x86)\National Instruments\LabVIEW 2012\vi.lib\Motion\FlexMotion\FunctionsVIs\Trajectory.llb\Load Acceleration/Deceleration.flx I have created new copies of these VIs with new names and included them in the MICAS-X NI Motion Driver sub-vis directly.

8 Directions for Adding a Configuration Parameter

Often, an addition of functionality or a bug fix to a Driver, Instrument, or Display can require additional configuration parameters for the component. To add configuration parameters, follow the steps below. Note that these steps are written with respect to adding a simple paramter, such as a device address or a loop time numeric. Many parameters will require additional steps and logic. This includes any parameters such that their value affects the allowed values of other parameters. For these situations, additional logic will need to be added to ensure that when one parameter changes value, all the others are coerced to acceptable values. Another special case is for channel names. On the user interface, channel names are presented as ring controls, so that the user can simply select a channel from a list of existing channels. But channel names are saved to the configuration file as strings. (If they were saved as rings (e.g. numbers), the position of a specific channel in the list of all channels could easily change, and the ring would be interpreted incorrectly subsequently.) Some additional hints for handling channel names are given at the bottom of this section. However, for both channel names and other complex situations, it is usually best to review how other MICAS-X editors handle the parameters and model your logic after that.

- Open the Editor for the component by double clicking on it in the Editors folder of the project.

- Open the block diagram and navigate to the Read File case. Double click on the MICAS Read xxx Parameters.vi icon to open the sub-vi that reads the configuration parameters.

- On the front panel of the Read Parameters VI, highlight the Parameters cluster type-def by clicking on its border. Right click on the border and select “Open Type Def.”

- On the type-def, add the new parameter. Make sure to give it a useful label and description. Highlight the parameter and press CTRL-C to copy it to the clipboard.

- Close and save the type-def.

- Close and save the Read Parameters VI.

- On the Editor VI front panel, press CTRL-V to paste the new parameter onto the front panel. Make sure it is a control, not an indicator.

- On the Editor's block diagram, find the terminal of the new parameter, right click and select “Create\Local Variable”.

- Move the local variable into the Display case.

- Expand the Unbundle by Name element so that the new parameter is available, and wire it to the local variable.

- Note that the Display Special case is used by some editors but not others. The purpose of this case is to display parameters that change values depending on the new values of other parameters. Review other Drivers for examples if needed.

- Go to the Wait case, and within the event structure find the last event. Right click on the event structure and select “Duplicate Event. Case”

- In the resulting dialog, select the new control as the Event Source and make sure that Value Change is selected as the Event.

- In the new event case, delete the control that was automatically duplicated from the previous event case. Move the new control into the case and wire it to the Bundle by Name node. Select the proper parameter within the Bundle by Name node to match the control. E.g. this control's value should be bundled into its corresponding element of the Parameters cluster.

- In the upper right corner of the new event case, make sure that there is an Enqueue node with the “Enable S & R” command. This will cause the Save button to flash when the parameter has been updated. If changes to this parameter can cause automatic changes to other parameters, then there should also be an Enqueue node added that sends the editor to the Display case.

- Save the Editor.

- Open the related Driver, Instrument, or Display. Implement the new parameter as needed.

- If the new parameter is a setting that the user can alter when the program is running, then paste a copy of the new control onto the front panel of the

Instrument or Display. In the Initialize case, unbundle the value of the parameter from the configuration and use a local variable for the new control to initialize it to the configuration value. Use the control terminal in an event structure or elsewhere as needed.

For channel name parameters, keep the following in mind: The parameter defined in the type def cluster should be a simple string, such that the configuration file stores the actual channel name as text. However, on the Editor front panel, the parameter should be a ring control, so that the user can select a channel name from a list of known channels. Right click on the ring control and create a property node with the property “Strings”. Use the Channels list to populate that ring control in the Initialize case of the Editor. Also, set the value of the ring (e.g. which channel is selected in the ring) in the Display case by using the “MICAS Find Name.vi” function to look up the string name from a list of all names. Again, refer to an existing Editor for examples.

9 Controllers Which Must Also Indicate

A common situation in the MICAS-X program as well as in Display modules is when a Controller (output) channel must be made available to the end-user, so that the operator can change the value, but that same LabVIEW control must reflect any changes that might be made to the Controller channel elsewhere in the MICAS-X system. E.g. a setpoint channel may be changed within a Sequence, or on a Display panel. The value in the control on the Display must reflect any changes made at any time by the Sequence, as well as any changes made directly by the user. Without a signaling mechanism in place, a common problem is that when the user changes the Controller value on the UI, the UI will often be updated with the old value of the Controller channel, then again with the new value. So changing a channel from 0 to 1 will often result in it flashing back to 0 briefly and then settling in to 1. The reason for this is that it can take a fraction of a second for the command which sets it to its new value of 1 to get to the proper Driver, and then for the Driver to update the buffer of most recent data values. Meanwhile, the UI updates and sees the old value. On a subsequent update of the UI, the new value has become available, and it updates again.

Previous to version 1.4.6, a one-deep Controller Update queue was used to try to handle this, but this mechanism was imperfect, and some Controller channel UI elements could become out of sync with the actual values of the channel. As of version 1.4.6, a new mechanism has been introduced. This mechanism relies on a sub-vi named “MICAS Wait For New Value.vi”. Whenever a UI element is changed that changes a Controller Channel value, the “MICAS Wait For New Value.vi” should be called. This VI polls the data buffer every 50 ms until it finds that the data buffer contains the new value of the Controller Channel that has been set. In case something goes wrong or take even longer, the VI will time-out after 2 seconds. This VI prevents the UI from updated with the old value, since any updates will occur after the value has been propagated through the entire MICAS-X system (barring a time-out.) This VI should be called from any Display in

MICAS-X which has Controller channels directly available to the end-user. Note that Switches and Sequences, which do change the values of Controller Channels, do not need this mechanism, since they are typically overlaid with an invisible Boolean button. The Boolean button requests the change of state of the Switch Channel or the Sequence, but the indicator displaying the state of the Switch or Sequence is below the invisible button and is updated independently.

10 Boolean (True/False) Values

All data channels in MICAS-X are stored as double-precision reals. Channels that are inherently Boolean (e.g. have only two values, True and False) are typically stored as 1 for True, and 0 for False. The generally accepted rule for such data is that any non-zero value is interpreted as True. Thus when programming with such channels, it is tempting to use the <>0 comparison to convert a channel's double-precision value into its Boolean value. However, a special case occurs within MICAS-X due to the use of the value "NaN" (Not a Number) to represent missing data. When the <>0 operation acts on the value "NaN", the result is True. This is technically correct according to the IEEE standard, but results in unwanted behavior in MICAS-X. For instance, a MICAS-X condition (in a Sequence or Trigger) could be testing for a Boolean True value, so that an action is taken when a condition becomes ready. If the channel being checked starts with a value of "NaN" (as all channels do before their Drivers are fully initialized), the <>0 comparison would yield a True value when most likely a False would be more appropriate. To handle this case in Sequence and Trigger conditions, special conditions have been implemented, including =NaN, <>NaN, and <>NaN,0.

When channels are being displayed as Boolean values on a user interface, additional care must be taken. In this case, use the "MICAS Channel to Boolean.vi" sub-VI located in the Common directory to convert a channel value to its Boolean equivalent. This sub-VI contains logic for ensuring that NaN is not interpreted as True.