

CMC Driver Framework Core Manual

by Choose Movement Consulting, LLC

1010001-01-01 REV A

Contents

1.	General Information	2
1.1.	User Manual Content.....	2
1.2.	Introduction	3
1.2.1.	General Description	3
1.2.2.	Product Support	3
1.3.	System Requirements	3
1.4.	End User License Agreement	4
1.5.	Referenced Documents.....	4
1.6.	Acronyms, Definitions, and Abbreviations.....	4
1.6.1.	Acronyms	4
1.6.2.	Definitions	4
1.6.3.	Abbreviations	4
2.	Software Installation.....	4
2.1.	First Time Install.....	4
2.2.	Subsequent Installations.....	5
2.2.1.	CMC Driver Framework Installer.....	5
2.2.2.	NI Package Manager	5
3.	Software Activation.....	5
4.	Software Component Descriptions.....	7
4.1.	CMC Configuration Manager	7
4.1.1.	Menu Options	8
4.1.2.	Configuration Settings.....	12
4.1.3.	User Configurations	12
4.1.4.	Status Indicator	12
4.2.	Console.....	13
4.2.1.	Menu Options	14
4.2.2.	Refresh	17

4.2.3.	Instrument Selector	17
4.2.4.	Console Log	17
4.2.5.	Injector Message	18
4.2.6.	Send Message	18
4.2.7.	Console API	18
4.3.	Instrument Control	19
4.3.1.	API	20
4.3.2.	Classes	22
5.	Tutorials	27
5.1.	CMC Configuration Manager	27
5.1.1.	TCP Listener Demo VI	28
5.1.2.	TCP Client Demo VI	29
5.1.3.	Configuration	30
5.1.4.	Run Demo	32
5.2.	Console Tutorial	33
5.2.1.	Viewing Console Logs	33
5.2.2.	Injection	33
5.3.	TCP Tutorial	34
5.3.1.	Creating UI	35
5.3.2.	Creating Processor	44
5.3.3.	Running UI and Processor	51
6.	Troubleshooting Recommendations	52
6.1.	Console Refuses to Inject into a connection that has active communication.	52
6.2.	Additional Resources.	52

1. General Information

1.1. User Manual Content

This user manual covers the description, system requirements, installation instructions, operating instructions, tutorials, and suggested troubleshooting tips for the CMC Driver Framework Core software.

1.2. Introduction

1.2.1. General Description

The CMC Driver Framework is an advanced framework for developing and implementing hardware abstraction layers in LabVIEW. It consists of:

- A) The Configuration Manager: A graphical user interface used to create instrument and simulated instrument configurations.
- B) The Console: A graphical user interface that may be used to monitor, record, and inject communication between the PC and instrument.
- C) The Instrument Control API: A set of VIs used to abstract communication interfaces (i.e. Serial, TCP).
- D) Device Type APIs: A set of VIs used to abstract specific device types (i.e. DC Power Supply).

Using these tools, a developer can implement instrument automation code once and change device or communication interface by selecting a new configuration file. The CMC Driver Framework Core is the foundation for the CMC Driver Framework. It includes the Configuration Manager, Console, and Instrument Control API.

1.2.2. Product Support

For technical support:

- Post to the CMC Driver Framework community located at <https://forums.ni.com/t5/CMC-Driver-Framework/gp-p/5394>.
- Email support@choose-mc.com.

It is encouraged to use the CMC Driver Framework community when possible so that others can learn from your questions.

1.3. System Requirements

System requirements match the requirements for the version of LabVIEW that the CMC Driver Framework is being installed for. For more details, see: <https://www.ni.com/en-us/support/documentation/supplemental/17/system-requirements-for-labview-development-systems-and-modules.html>.

The CMC Driver Framework also requires the software in the following table. With the exception of LabVIEW, the software modules will be installed or upgraded as part of the installation process.

Software Requirement	Minimum version
LabVIEW Development System	2017 SP1
NI-LabVIEW Command Line Interface x86	2.1
NI-488.2 Runtime	19.5
NI-Serial Runtime	17.5
NI-VISA Runtime	18.0

1.4. End User License Agreement

Copyright © 2016, Choose Movement Consulting, LLC. All rights reserved. This software is part of the CMC Driver Framework. By using this software, you agree to the CMC Driver Framework and CMC Driver End User License Agreement that is found at <https://choose-mc.com/pages/legal>.

1.5. Referenced Documents

Document	Source
CMC EULA	https://choose-mc.com/pages/legal
LabVIEW System Requirements	https://www.ni.com/en-us/support/documentation/supplemental/17/system-requirements-for-labview-development-systems-and-modules.html

1.6. Acronyms, Definitions, and Abbreviations

1.6.1. Acronyms

Acronym	Meaning
CMC	Choose Movement Consulting, LLC
EULA	End User License Agreement
UI	User Interface
VI	Virtual Instrument
API	Application Programming Interface
IO	In/Out

1.6.2. Definitions

Term	Definition
.instconfig	The file extension used by the Driver Framework to identify file that contain all the necessary files to configure an Instrument.
Configuration File	Refers to an .instconfig
Driver Framework	Refers to the CMC Driver Framework.

1.6.3. Abbreviations

None.

2. Software Installation

The first installation of the CMC Driver Framework should use the Driver Framework Installer.exe. After the first time the installer is run, there are additional methods available - described below - for initiating installation of Driver Framework components. The first time through, the support files and runtime engines are installed. Subsequent Driver Framework component installations will use the support files.

2.1. First Time Install

1. Run Driver Framework Installer.exe. It is found in the ZIP file download for the CMC Driver Framework. Be sure to expand the ZIP file before running the installer.
2. Follow prompts to select which Driver Framework components to install.

2.2. Subsequent Installations

For subsequent installations, you can either use the CMC Driver Framework Installer.exe or NI Package Manager. See instructions below.

2.2.1. CMC Driver Framework Installer

1. Run Driver Framework Installer.exe from the folder you expanded it to.
2. Follow prompts to select which Driver Framework components to install.

2.2.2. NI Package Manager

1. Click on the Start Menu icon.
2. Scroll down and expand the "National Instruments" folder.
3. Select "NI Package Manager"; it might be necessary to run it with elevated privileges depending on your computer's settings.
4. Confirm that you want to allow NI Package Manager to make changes to your system.
5. When NI Package Manager has finished updating the feeds, select a tab as follows:
 - a. UPDATES: to install a newer version of an already installed API or Driver.
 - b. INSTALLED: to uninstall an already installed API or Driver.
 - c. PACKAGES: to install either a new API or Driver, or an API or Driver to a new version of LabVIEW.
6. Select the check box next to each version of an API or Driver to be installed, updated or uninstalled and click the button that will initiate your selection.
7. Follow the prompts from NI Package Manager to finish the action.

3. Software Activation

The CMC Driver Framework Core software is free and does not require activation. To activate licensed CMC Driver Framework add-ons, complete the following steps:

1. Open the version of LabVIEW the CMC Driver Framework was installed to.
2. Go to the **Tools** menu and select **CMC Tools >> License Manager...**

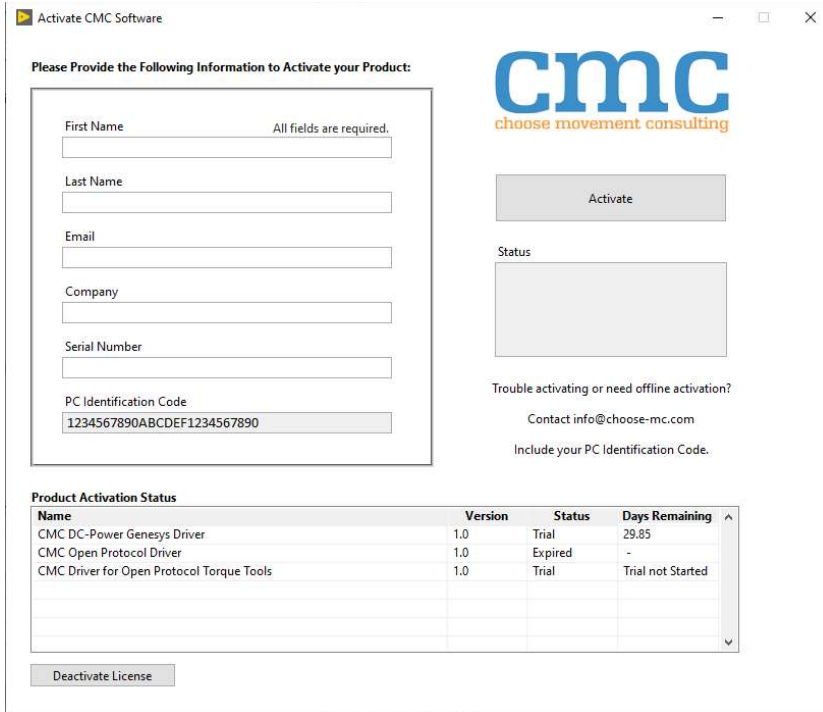


Figure 1: Activation Window

3. Enter your information in the respective fields. Serial Number is typically sent to the purchaser automatically upon purchase.
4. Click **Activate**.

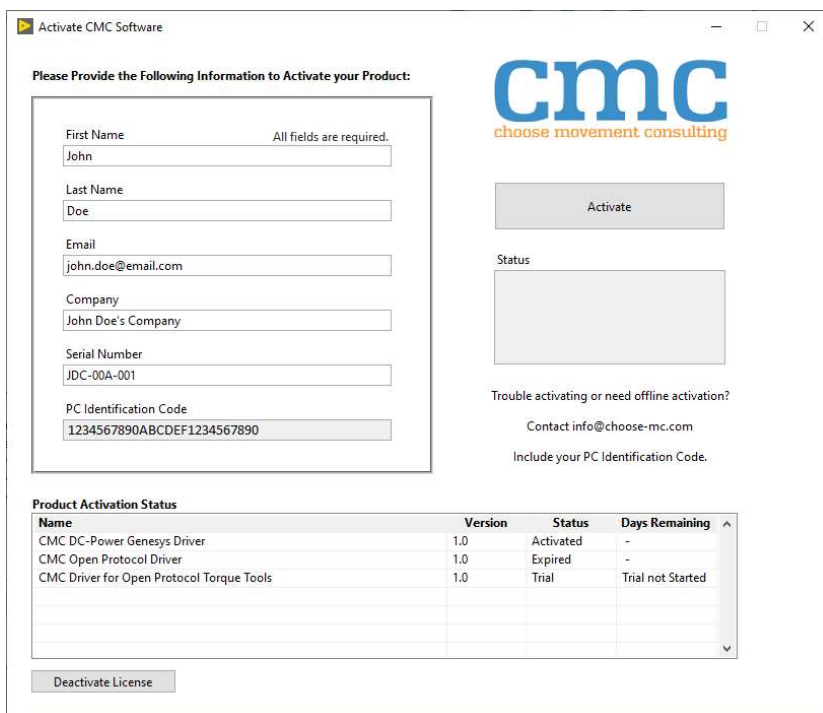


Figure 2: Software Activated

- For offline activation, copy the PC Identification Code and send it along with your request to support@choose-mc.com or info@choose-mc.com.

4. Software Component Descriptions

4.1. CMC Configuration Manager

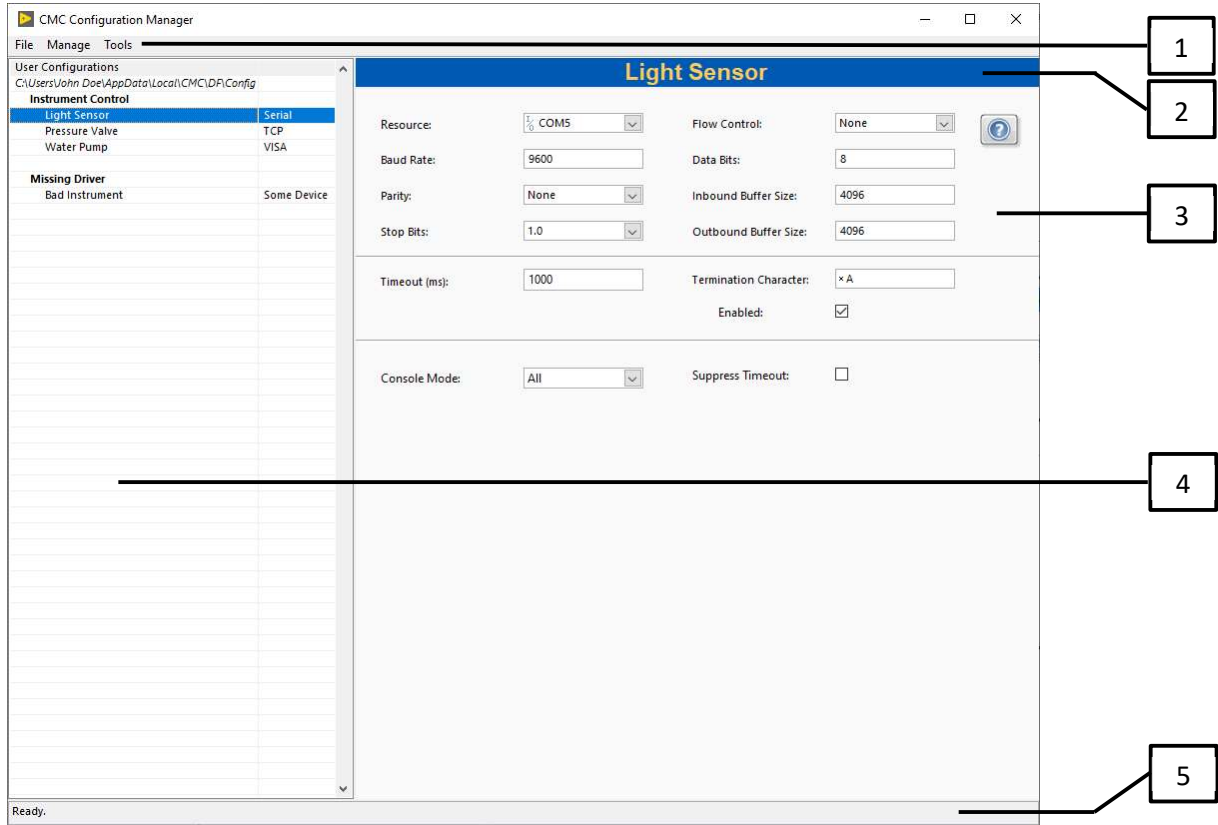


Figure 3: CMC Configuration Manager user interface

The CMC Configuration Manager provides a user interface that enables the creation of configuration files (.instconfig) for a supported instrument. These .instconfig files are used by the various Driver Framework APIs to manage the configuration of the instrument and communication with the instrument. For a tutorial on using the CMC Configuration Manager, please refer to Section 5.1.

Number	Name	Description	Section
1	Menus	May be used to access to the various actions the CMC Configuration Manager can perform.	4.1.1
2	Configuration Name	Displays the name of the currently selected configuration.	4.1.2
3	Configuration Settings	Provides access to all configuration items for the specific configuration. These will change according to the instrument type selected.	NA
4	User Configurations	Lists of all .instconfig configurations that have been found on the system.	4.1.3

Number	Name	Description	Section
5	Status Indicator	Displays user feedback on the status of the CMC Configuration Manager.	4.1.4

4.1.1. Menu Options

The CMC Configuration Manager has two menus in its menu bar: File and Manage.

4.1.1.1. File Menu

The File menu handles the following operations:

- Create New
- Save
- Save All
- Save For New Hardware (Not available for this release)
- Exit

4.1.1.1.1. Create New

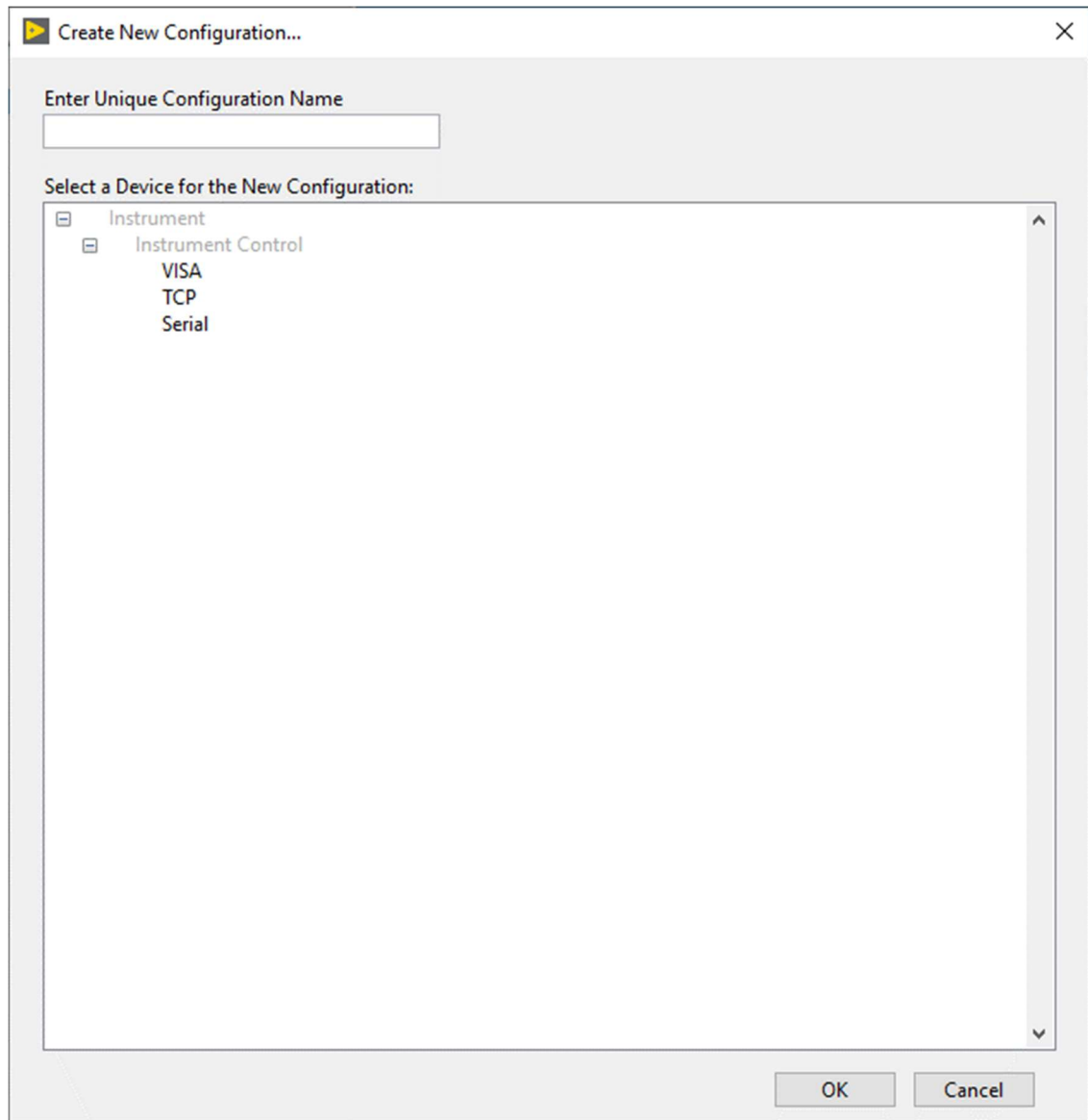


Figure 4: Create New Configuration Dialog

Selecting this menu option will launch the Create New Configuration dialog. The Create New Configuration dialog provides a user interface that enables the creation of a new configuration file. New configurations must have a unique name and an instrument type selected. Clicking OK will launch a file explorer dialog that the user may use to specify the file location where the configuration file will be saved. The new configuration is then created and selected in the CMC Configuration Manager.

4.1.1.1.2. Save

Saves the currently selected configuration.

4.1.1.1.3. Save All

Saves all the configurations.

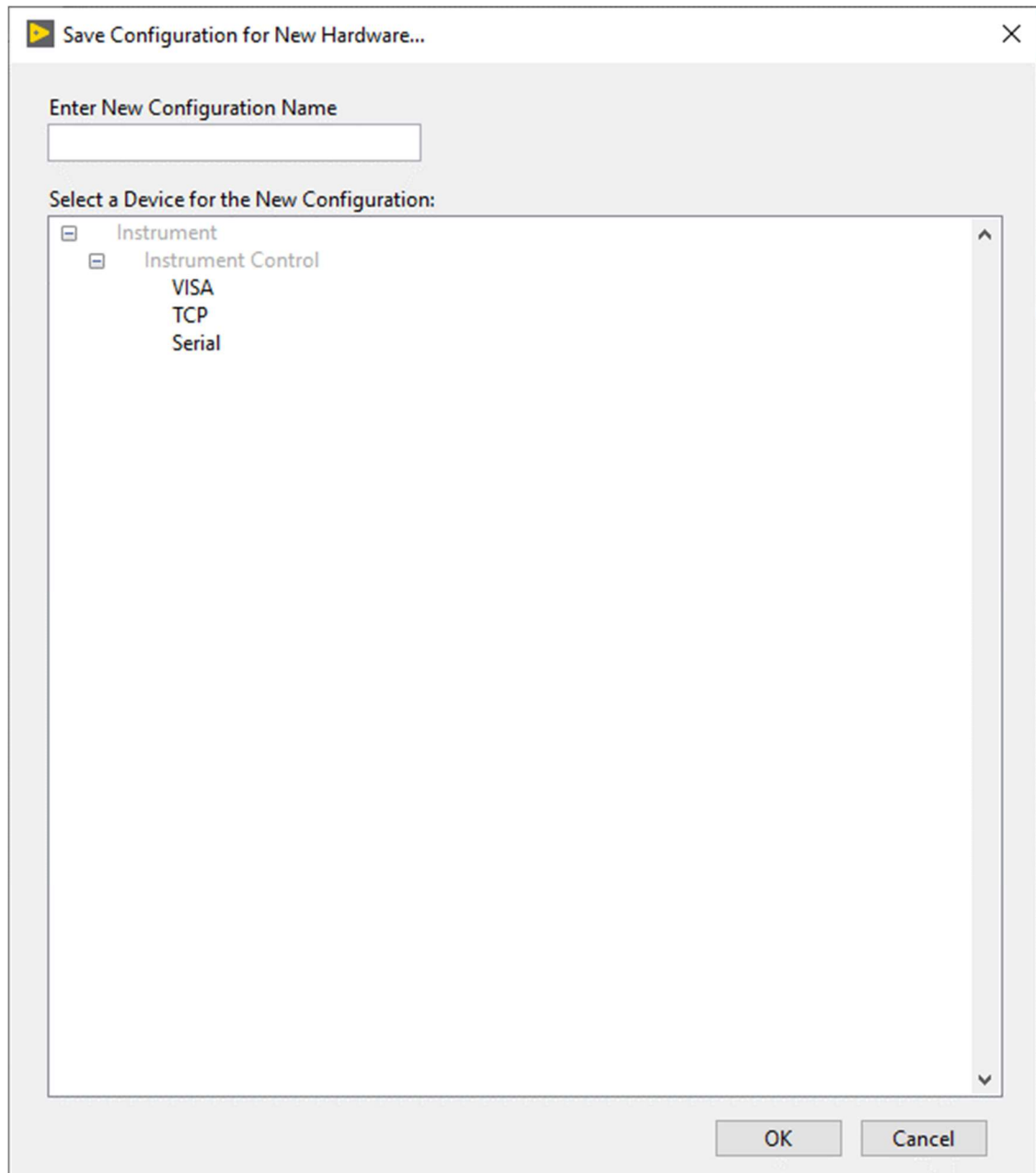


Figure 5: Save for New Hardware Dialog

This feature is not available in the current release.

Selecting this menu option while a configuration is selected will launch the Save for New Hardware dialog. The Save for New Hardware dialog provides a user interface for the creation of a new configuration file using an existing configuration file for the initial values. The existing configuration and new configuration do not need to be of the same instrument type or class. If they are both the same type, then all the configuration settings will be copied. If they are both the same class, then the settings supported by both will be copied. If they are neither the same class, nor type, then only the

instrument control settings will be copied, provided both instruments use an instrument control.

4.1.1.1.5. Exit

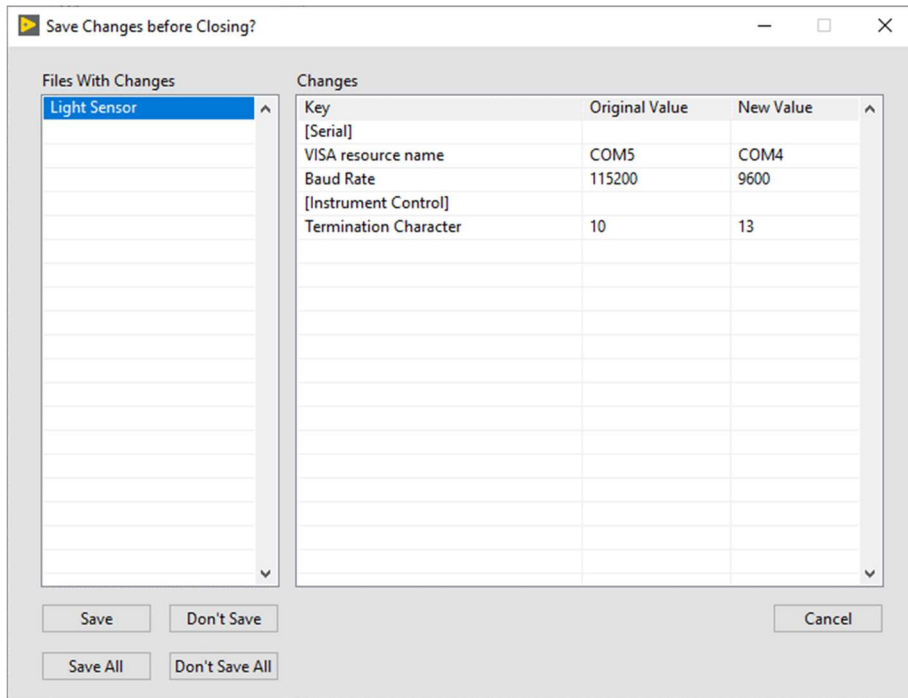


Figure 6: Save Changes Dialog

When this menu option is selected, the CMC Configuration Manager will check if any configuration files have unsaved changes. If there is at least one unsaved change, it will launch the Save Changes dialog.

Name	Description
Files With Changes	Displays all the configuration files with unsaved changes. Selecting an item in this list will display the unsaved changes in the Changes list.
Changes	Displays all the unsaved changes in the selected file. These are organized by the section (in brackets). Each change is listed as its name (Key), original value and the new value.
Save	Saves the currently selected File With Changes. If this is the last file, the dialog closes and the CMC Configuration Manager will close.
Discard	Discards the all the unsaved changes in the currently selected File With Changes. If this is the last file, the dialog closes and the CMC Configuration Manager will close.
Save All	Saves all the Files With Changes. The dialog then closes and the CMC Configuration Manager will close.
Discard All	Discards all the unsaved changes. The dialog then closes and the CMC Configuration Manager will close.

Name	Description
Cancel	Cancels the exit. All unsaved changes will remain unsaved. They will not be discarded or saved.

4.1.1.2. *Manage Menu*

The Manage Menu handles the following operations:

- Refresh
- Import .instconfig
- Delete Configuration

4.1.1.2.1. Refresh

Refreshes the User Configurations list.

4.1.1.2.2. Import .instconfig

Opens a file explorer window that the user may use to navigate to an .instconfig file. Once a file is selected, it will be copied to the default configuration directory.

4.1.1.2.3. Delete Configuration

Deletes the currently selected configuration.

4.1.2. Configuration Settings

This displays the name of the selected user configuration. Beneath the name, the instrument's configuration interface is displayed. Since these interfaces vary by instrument, see the instrument's respective user manual or help dialog for more information on the items here.

4.1.3. User Configurations

The User Configurations list displays all the .instconfig files found in the system. These files are grouped together according to file location (*italics*), then instrument class (**bold**), then by instrument type (second column) and finally alphabetically.

The User Configurations supports a right-click menu with the following options:

Name	Description
Save	Saves any changes to the currently selected configuration.
Delete	Prompts the user if they are certain they would like to delete the selected configuration. If the user responds positively, then the selected configuration is deleted.

4.1.4. Status Indicator

Displays what the manager is currently doing. Most actions will update this with appropriate information. Once an action is completed, this will display "Ready".

4.2. Console

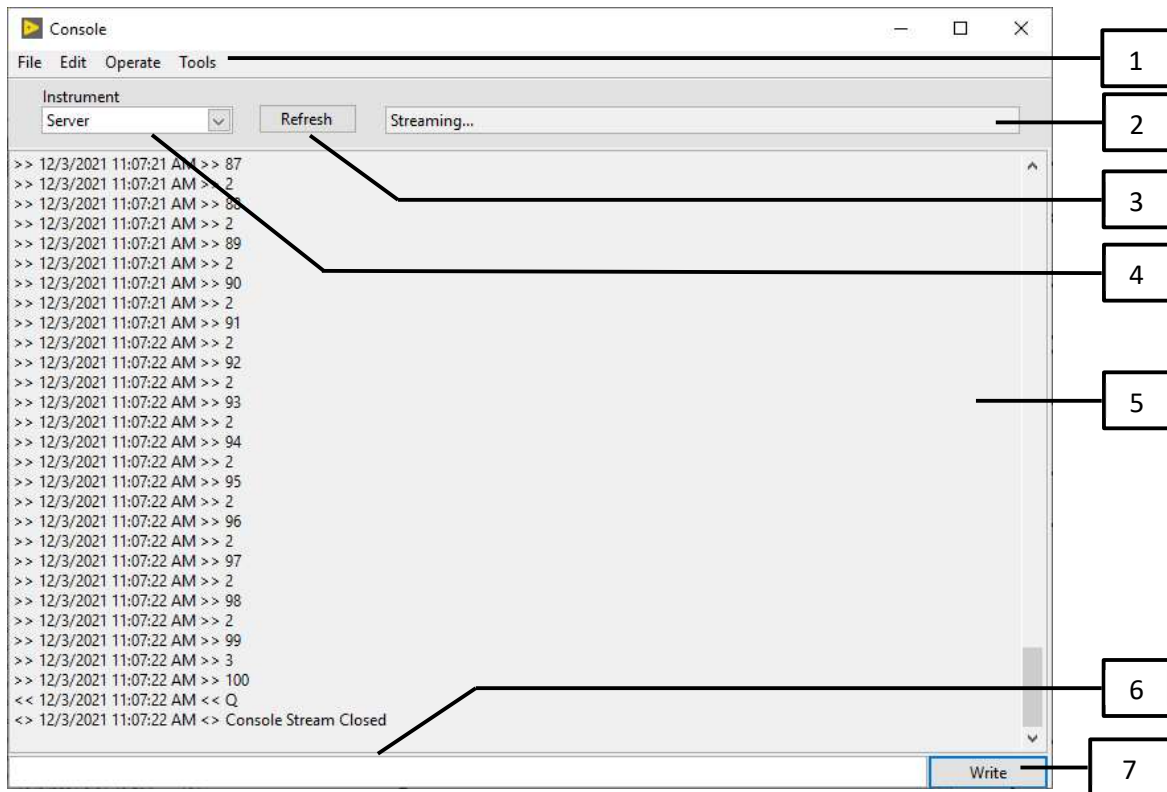


Figure 7: Console User Interface

The Console provides an interface that may be used to monitor communication between Driver Framework Instruments and the host computer. These communications are saved to a text file for later review. The Console may also be used to send commands and queries directly to the selected instrument. The Console runs as a background process whenever a Driver Framework Class is used. Its UI can be launched through the Instrument Control API Initialize VI (CMC >> Instrument Control Palette) or the Launch Console VI (CMC >> Console Palette).

The Console requires the VIs writing to it to be inside a LabVIEW project in order for the Console to properly log the data to an output.

Number	Name	Description	Section
1	Menus	Provides access to the various actions the Console can perform.	4.2.1
2	Status Display	Displays what the Console is doing.	NA
3	Refresh	Refreshes the list of instruments available in the Instrument Selector.	4.2.2
4	Instrument Selector	Provides an interface that may be used to select which instrument's IO to display.	4.2.3
5	Console Log	Displays the selected instrument's IO.	4.2.4
6	Injector Message	Provides an interface that may be used to inject messages into an instrument's IO.	4.2.5

7	Send Message	Sends the Injector Message to the selected instrument.	4.2.6
---	--------------	--	-------

4.2.1. Menu Options

The Console has four following menus in its menu bar:

- File
- Edit
- Operate
- Tools

4.2.1.1. File Menu

The File Menu handles the following operation:

- Exit

4.2.1.1.1. Exit

Closes the console UI. Any closed instruments are removed from the available instruments and complete log files are generated. Log files are placed at C:\Users\\AppData\Local\CMC\DF\Console by default. Log files are named in the following format: <ConsoleName>-mm_dd_yyyy_hh_mm_ss_AM|PM.txt.

4.2.1.2. Edit Menu

The Edit Menu handles the following operations:

- Undo
- Redo
- Cut
- Copy
- Paste

4.2.1.2.1. Undo

Reverts the previous action. Note that some actions, like instrument injection cannot be undone.

4.2.1.2.2. Redo

Redoes an action that had been undone previously.

4.2.1.2.3. Cut

Copies the selected text to the clipboard and deletes it.

4.2.1.2.4. Copy

Copies the selected text to the clipboard.

4.2.1.2.5. Paste

Copies the text in the clipboard into the selected position of a control.

4.2.1.3. Operate Menu

The Operate Menu handles the following operations:

- Refresh
- Abort Transmission
- Injection Mode

4.2.1.3.1. Refresh

Checks for updates to the active instruments. Any new instruments are added to the selector.

4.2.1.3.2. Abort Transmission

Aborts the current transmission that is taking place with the selected instrument. Note that this can introduce errors in the selected instrument and should be used with care.

4.2.1.3.3. Injection Mode

Provides a way for the user to select how the injected message will be handled. Two options are provided:

- **Query:** Sends a message and collects a response.
- **Write Only:** Sends a message, but does not look for a response.

For more details, see Section 4.3.1.3.

4.2.1.4. Tools Menu

The Tools Menu handles the following operations:

- Options...
- Help

4.2.1.4.1. Options...

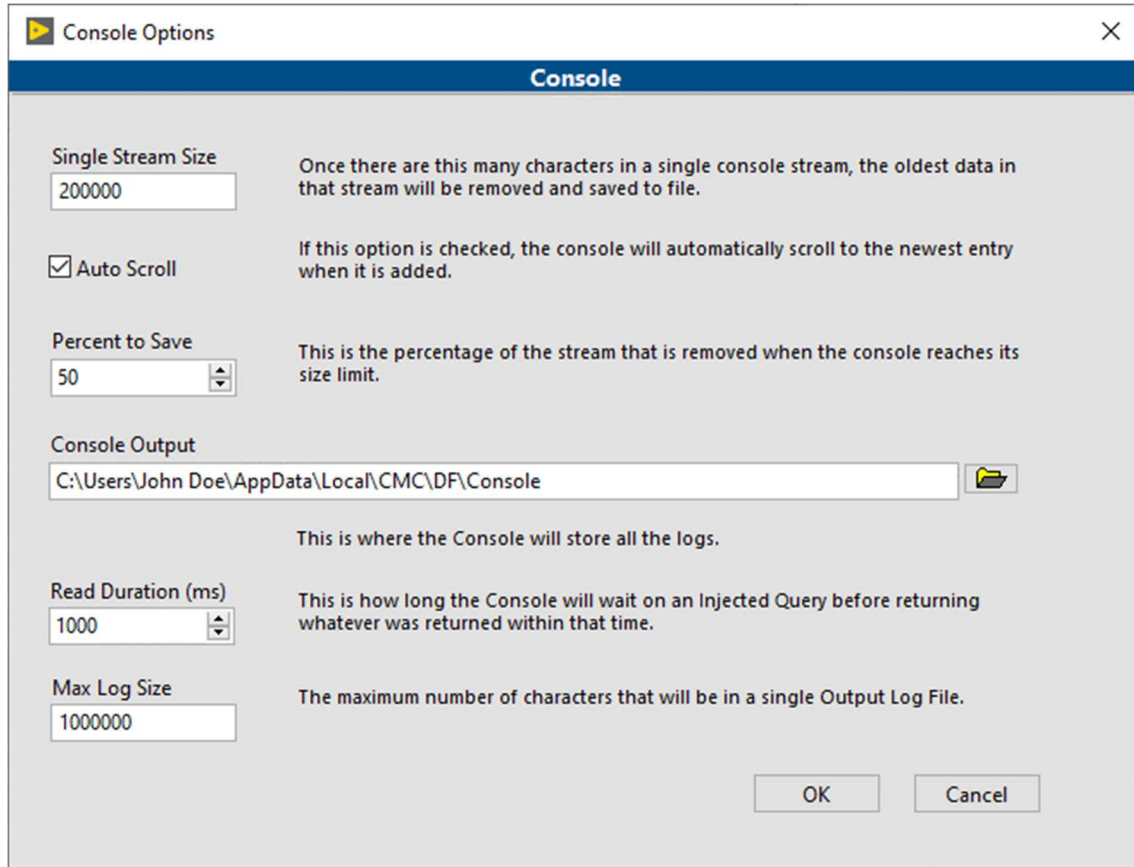


Figure 8: Console Options window

Selecting this menu option launches the Console Options window. This window may be used to customize the Console behavior.

The Options window provides the following options:

Name	Description
Single Stream Size	This defines the maximum characters a single instrument can have in the console at a time. Once this value is exceeded, the Console removes the oldest data to prevent excessive memory use.
Auto Scroll	When this option is checked the Console automatically scrolls to the newest entry when it is added.
Percent to Save	This is the percentage of the Console data that is removed when the size is exceeded. Allowed values are 1-100%.
Console Output	This defines where the Console will store the log files that it generates.
Read Duration (ms)	This defines how long the Console will wait, in milliseconds, for an Injected Query to return data. Once the Duration has completed, any collected data is returned.

Name	Description
Max Log Size	The maximum number of characters that can be stored in a single log file. If the log exceeds this size, it will create an additional log file to store additional content.
OK Button	Saves the current Options and applies them to the Console.
Cancel Button	Discards any changes to the Console settings.

4.2.1.4.2. [Help](#)

Opens this document.

4.2.2. Refresh

Updates the list of instruments available in the Instrument Selector.

4.2.3. Instrument Selector

This dropdown will show all the instruments currently registered in the Console. The selected instrument has its IO messages displayed in the Console Log.

4.2.4. Console Log

Displays all the communication between the host computer and the selected instrument. There are four type of messages that can be displayed. The Console Log supports a right-click menu that enables the user to toggle the auto scrolling feature. The Console Log also has a limited size to prevent excessive memory usage, once this size is exceeded, the oldest data is removed and saved to file.

For more details on the Console Log options, see Section 4.2.1.4.1.

4.2.4.1. *Incoming Message*

Incoming messages are messages that have been sent to the selected instrument. These are displayed in the following format: >>Timestamp>>Message.

4.2.4.2. *Outgoing Message*

Outgoing messages are messages that have been received from the selected instrument. These are displayed in the following format: <<Timestamp<<Message.

4.2.4.3. *Comment Message*

Comment messages are simply useful information that is sent to the Console, but is not sent to any instrument. These messages are displayed in the following format: <>Timestamp<>Message.

4.2.4.4. *Error Message*

Error messages are generated when the Write to Stream function receives an incoming error that is not suppressed. These are displayed in the following format: >!<Timestamp>!<Message

4.2.5. Injector Message

This enables the user to type the text to send to the instrument. This supports a right-click menu that provides a way for the user to change the input display type between normal text, backslash codes and hexadecimal codes.

4.2.6. Send Message

Clicking this button will send the text in the Injector Message to the selected instrument. This can operate in two different modes, which are controlled by the Write to Instrument menu option, see Section 4.2.1.3.3 for more details.

Name	Description
Write	In write mode, the button will display 'Write'. Clicking the button in this state will send a message to the selected instrument.
Query	In query mode, the button will display 'Query'. Clicking the button in this state will send a message to the selected instrument and read back a response.

4.2.7. Console API

The Console API provides direct access to some of the console's functions.

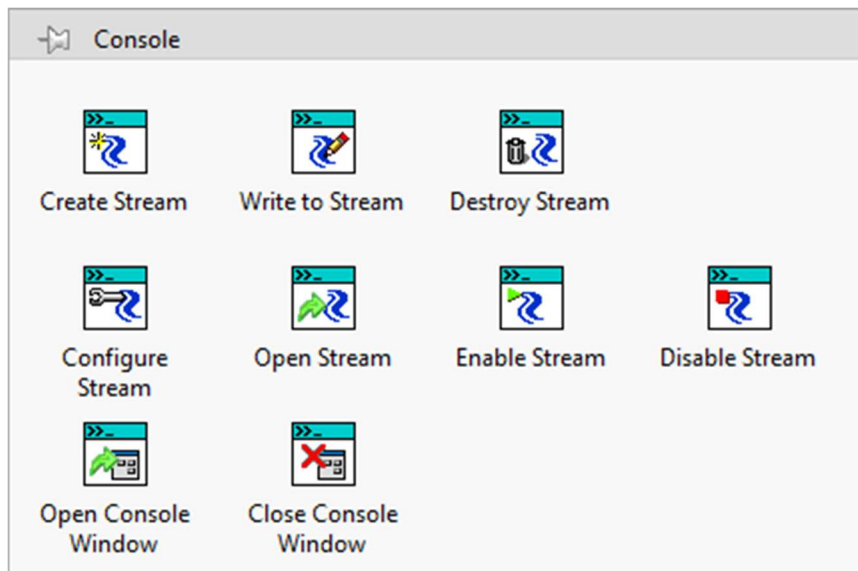


Figure 9: Console API LabVIEW Palette

4.2.7.1. Create Stream

Creates a Console Stream with the provided Name. If the Console is not already running, it is launched. If an instconfig is provided, it will also configure the Stream. If Open Console Window is set to True, it will open the Console's User Interface.

4.2.7.2. Write to Stream

Writes the provided message to the associated Stream if Console is set to True and the Mode is Selected. If the Mode is All, the message will also be written to the Stream. Any other configuration of Console and Mode will not write to the Stream.

This function will also write error messages to the Stream, provided that there is an unsuppressed error and the function would be able to write to the Stream if there wasn't an incoming error.

4.2.7.3. Destroy Stream

Destroys the Stream associated with the provided Console, or the provided name.

4.2.7.4. Configure Stream

Configures the Stream according to the provided settings file. These settings pertain to Error Suppression and the initial Mode.

4.2.7.5. Open Stream

Informs the Console that the Stream is open. Streams must be opened before injection is allowed. Writing to an unopened Stream is still permitted in most cases.

4.2.7.6. Enable Stream

Changes the Stream's Mode to All, so that any message provided will be written to the Stream.

4.2.7.7. Disable Stream

Changes the Stream's Mode to None, so that any message provided will not be written to the Stream.

4.2.7.8. Open Console Window

Opens the Console's User Interface. If the Console is not already running, it will be launched.

4.2.7.9. Close Console Window

Closes the Console's User Interface. The Console will continue to run while its User Interface is closed. If there are no active Streams, the Console will stop executing.

4.3. Instrument Control

Instrument Control provides an abstraction layer for any supported Instrument IO operation. This enables the user to program using the same functions, regardless of the specific protocol that will be used for the communication.

4.3.1. API

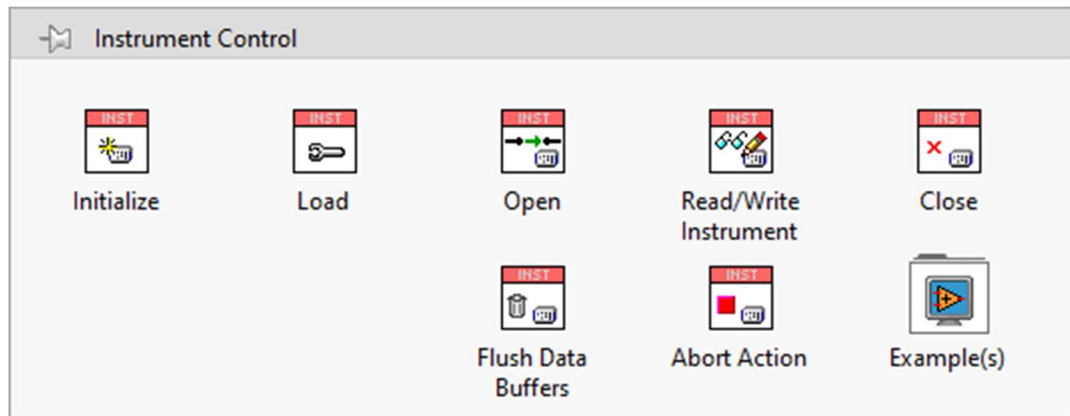


Figure 10: Instrument Control API LabVIEW Palette

The Instrument Control API is the functions provided on the LabVIEW Palette for programming with the Instrument Control.

4.3.1.1. *Initialize*

Initializes the Instrument Control. It returns a generic Instrument Control object. This also launches the Console and adds the Instrument Control to the Console (with the user provided name) so that the communication can be observed.

4.3.1.2. *Load Instrument*

Generates a specific Instrument Control object from the input.

4.3.1.2.1. *Load From Configuration File*

Takes a file path to the .instconfig file that the Instrument Control will be created from.

4.3.1.2.2. *Load From Configuration Tool*

This instance takes the name of the .instconfig file. Note that this is the name found in the .instconfig file as the Configuration_Name of the Instrument section, not the name of the file. This is the same name as it appears in the CMC Configuration Manager.

4.3.1.3. *Read Write Instrument*

Handles writing to and reading from the Instrument.

4.3.1.3.1. *Read Bytes from Instrument*

Reads a specified number of bytes from the Instrument.

4.3.1.3.2. *Read Until Data from Instrument*

Reads data from the Instrument until it reads a specified string.

4.3.1.3.3. *Query Instrument (Bytes)*

Sends a message to the instrument and then reads a specified number of bytes from the instrument.

4.3.1.3.4. *Query Instrument (Data)*

Sends a message to the instrument and then reads data from the Instrument until it reads a specified string from the instrument.

4.3.1.3.5. *Write to Instrument*

Sends a message to the instrument.

4.3.1.4. *Close Instrument*

Closes the instrument communication. If Destroy is TRUE (default), then the class is also reset to its default state.

4.3.1.5. *Flush Data Buffers*

Flushes the input and output buffers of the instrument.

4.3.1.6. *Open*

Opens the communication protocol.

4.3.1.7. *Abort Action*

Aborts the current communication action. This can be used locally, on the Instrument Control this is wired to, or applied to every Instrument Control.

4.3.2. Classes

4.3.2.1. TCP

TCP Configuration


IP Address:	<input type="text" value="127.0.0.1"/>	TCP Write Timeout (ms):	<input type="text" value="2000"/>	
Service Name:	<input type="text"/>	TCP Open Timeout (ms):	<input type="text" value="20000"/>	
TCP Port:	<input type="text" value="1234"/>	Connection Type:	<input type="text" value="Standard"/>	<input type="button" value="v"/>
Timeout (ms):	<input type="text" value="1000"/>	Termination Character:	<input type="text" value="× A"/>	
		Enabled:	<input checked="" type="checkbox"/>	
Console Mode:	<input type="text" value="All"/>	Suppress Timeout:	<input type="checkbox"/>	

Figure 11: TCP Configuration User Interface

Name	Description
IP Address	Specifies the IP address of the TCP Instrument to which this instconfig file is intended to communicate.
Service Name	Allows for a TCP session to be named.
TCP Port	The port over which the host computer and TCP instrument are intended to communicate.
TCP Write Timeout (ms)	Specifies the time, in milliseconds, that the TCP write command uses as the timeout before returning an error.

Name	Description
TCP Open Timeout (ms)	Specify the time, in milliseconds, that the Open command waits for a connection to the TCP instrument to be established before returning an error.
Connection Type	The type of TCP connection, default is Standard Connection. <ul style="list-style-type: none"> • <u>Standard Connection</u>: Creates a TCP Client. • <u>Create Listener</u>: Creates a TCP Listener.
Timeout (ms)	Specifies the read timeout, in milliseconds, normally used for waiting for completed messages to be returned from the TCP instrument. This timeout can be overridden by specific read operations, but by default this Timeout value will be used.
Termination Character	Specifies the character that defines the end of a data packet.
Enabled	Specifies whether the Termination Character setting is used or not.
Console Mode	Specifies how the send and receive messages interact with the console. <ul style="list-style-type: none"> • <u>All</u>: Sends all messages to and from the TCP instrument to the Console. • <u>Selected</u>: Enables the user to specify, in the software, which messages are written to the Console. • <u>None</u>: Does not send any messages to the Console.
Suppress Timeout	Specifies whether the Console Stream will suppress the Driver Framework Timeout Error code.
Help Button	Opens a help window that explains the other user interface items for a TCP configuration.

4.3.2.2. Serial

Serial Configuration

Resource: <input style="width: 80%;" type="text" value="COM8"/>	Flow Control: <input style="width: 80%;" type="text" value="None"/>	
Baud Rate: <input style="width: 80%;" type="text" value="9600"/>	Data Bits: <input style="width: 80%;" type="text" value="8"/>	
Parity: <input style="width: 80%;" type="text" value="None"/>	Inbound Buffer Size: <input style="width: 80%;" type="text" value="4096"/>	
Stop Bits: <input style="width: 80%;" type="text" value="1.0"/>	Outbound Buffer Size: <input style="width: 80%;" type="text" value="4096"/>	
Timeout (ms): <input style="width: 80%;" type="text" value="1000"/>	Termination Character: <input style="width: 80%;" type="text" value="× A"/>	
	Enabled: <input checked="" type="checkbox"/>	
Console Mode: <input style="width: 80%;" type="text" value="All"/>	Suppress Timeout: <input type="checkbox"/>	


Figure 12: Serial Configuration User Interface

Name	Description
Resource	The physical resource that is used for the serial connection.
Baud Rate	The amount of bits sent through the connection per second. Common baud rates are 4800, 9600, 19200 and 115200.
Parity	A method of detecting transmission errors. The serial instrument's documentation should specify this value; most commonly, this is None.
Stop Bits	Defines how many stop bits are expected.

Name	Description
Flow Control	Determines the type of handshaking (if any) that is done between the host computer and the serial instrument.
Data Bits	How many bits the data packet is composed of.
Inbound Buffer Size	The number of bits can be stored in the inbound buffer, waiting for the host computer to process them.
Outbound Buffer Size	The number of bits can be stored in the outbound serial buffer, waiting for the instrument to read them.
Timeout (ms)	Specifies the read timeout, in milliseconds, normally used for waiting for completed messages to be returned from the serial instrument. This timeout can be overridden by specific read operations, but by default this Timeout value will be used.
Termination Character	The character that defines the end of a data packet.
Enabled	Specifies whether the Termination Character setting is used or not.
Console Mode	<p>Specifies how the send and receive messages interact with the console.</p> <ul style="list-style-type: none"> • <u>All</u>: Sends all messages to and from the TCP instrument to the Console. • <u>Selected</u>: Enables the user to specify, in the software, which messages are written to the Console. • <u>None</u>: Does not send any messages to the Console.
Suppress Timeout	Specifies whether the Console Stream will suppress the Driver Framework Timeout Error code.
Help Button	Opens a help window that explains the other user interface items for a serial configuration.

4.3.2.3. VISA

VISA Configuration

Resource: 

Timeout (ms): Termination Character:

Enabled:

Console Mode: Suppress Timeout:

Figure 13: VISA Configuration User Interface

Name	Description
Resource	The physical resource that is used for the VISA connection. VISA supports a variety of interfaces such as GPIB, USB and LAN/LXI.
Timeout (ms)	Specifies the read timeout, in milliseconds, normally used for waiting for completed messages to be returned from the TCP instrument. This timeout can be overridden by specific read operations, but by default this Timeout value will be used.
Termination Character	Specifies the character that defines the end of a data packet.

Name	Description
Enabled	<ul style="list-style-type: none"> Specifies whether the Termination Character setting is used or not.
Console Mode	<p>Specifies how the send and receive messages interact with the console.</p> <ul style="list-style-type: none"> <u>All</u>: Sends all messages to and from the TCP instrument to the Console. <u>Selected</u>: Enables the user to specify, in the software, which messages are written to the Console. <p><u>None</u>: Does not send any messages to the Console.</p>
Suppress Timeout	Specifies whether the Console Stream will suppress the Driver Framework Timeout Error code.
Help Button	Opens a help window that explains the other user interface items for a VISA configuration.

5. Tutorials

5.1. CMC Configuration Manager

This tutorial will explain how to create configuration files (instconfig) that can be used in the included demo. The demo consists of a client and listener that communicate with each other through any supported protocol. As such, it includes several details about the demo (Sections 5.1.1, 5.1.2, and 5.1.4) as well as the steps required to create an instconfig file (Section 5.1.3).

The demo can be opened in the following ways:

Method	Instructions
LabVIEW example finder	<ol style="list-style-type: none"> Open LabVIEW. On the home screen, go to the Help Menu >> Find Examples... In the NI Example Finder, either: <ol style="list-style-type: none"> In browse, open Toolkits and Modules >> Third-Party Add-Ons >> Other. In search, type CMC and click search. Open Client.vi and Listener.vi.
LabVIEW Palettes	<ol style="list-style-type: none"> Create a new VI. Right-click on the block diagram. In the Palettes, locate CMC >> Instrument Control >> Examples. Open Client.vi and Listener.vi.

5.1.1. TCP Listener Demo VI

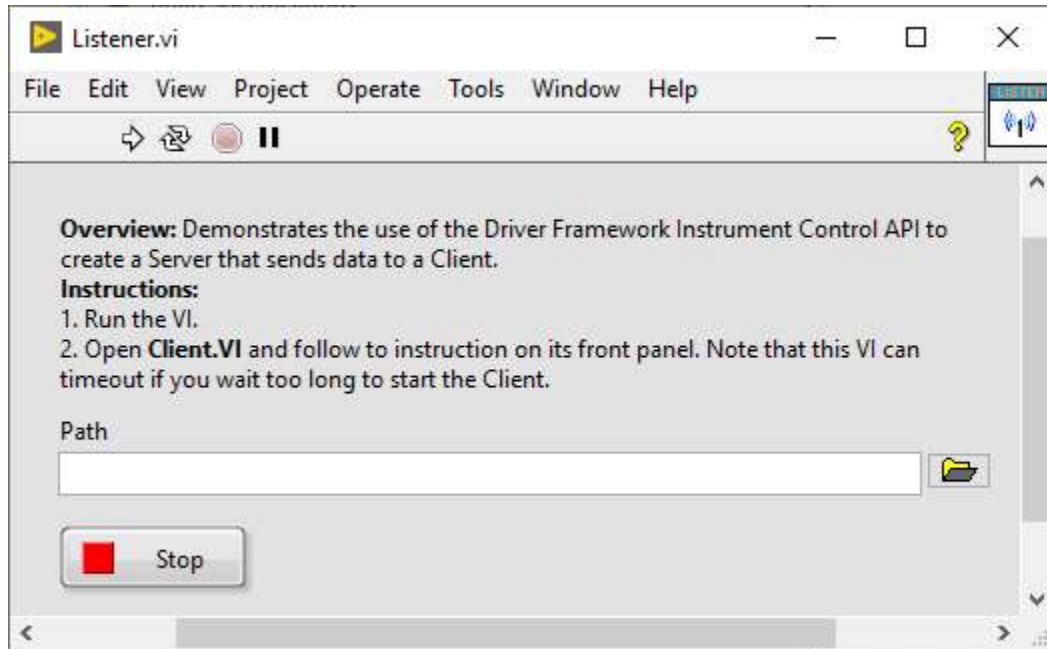


Figure 14: Listener Demo Front Panel

This is a simple Listener. It will send data to the Client. Once this Listener is initialized, it counts how many times it has looped and sends that number to the Client. It then checks for any Client messages. If there aren't any, it repeats. If there is a message from the Client, the Listener shuts down.

5.1.1.1. Initialize

First and foremost, the Listener will need to initialize an instrument control. The initialize function just needs a name that is unique and sufficiently descriptive. This is used to register the instrument control with the Console so that it is easier to keep track of.

5.1.1.2. Load Instrument

Now that an instrument control has been created, it needs to be loaded from file. This creates a specific instance of an Instrument Control object that the subsequent functions will use. For the sake of this tutorial, it will be assumed that it is a TCP instrument control.

5.1.1.3. Open

Opens the TCP connection. Since this is the Listener, it will create a TCP Listener and wait for a Client to connect to it. Once it has a Client, it will start looping through its Listener code.

5.1.1.4. Write to Instrument

The data the Listener sends to the Client comes in two packets. The first is a single character that tells the Client how many characters will be in the following data packet. The second is the number of times the Listener has looped.

5.1.1.5. Read Bytes

Once the Listener has finished communicating to the Client, it waits for a response from the Client for 50 milliseconds. If the Client responds, this is interpreted as a close prompt. However, if no response is received, the Listener will rerun the loop.

5.1.1.6. Close

Once the Listener has received a stop command, it tells the Console that the connection has been closed and closes the connections. However, the TCP instrument control is not destroyed, so it could be reopened later. After the connection has been closed, the Listener shuts down.

5.1.2. TCP Client Demo VI

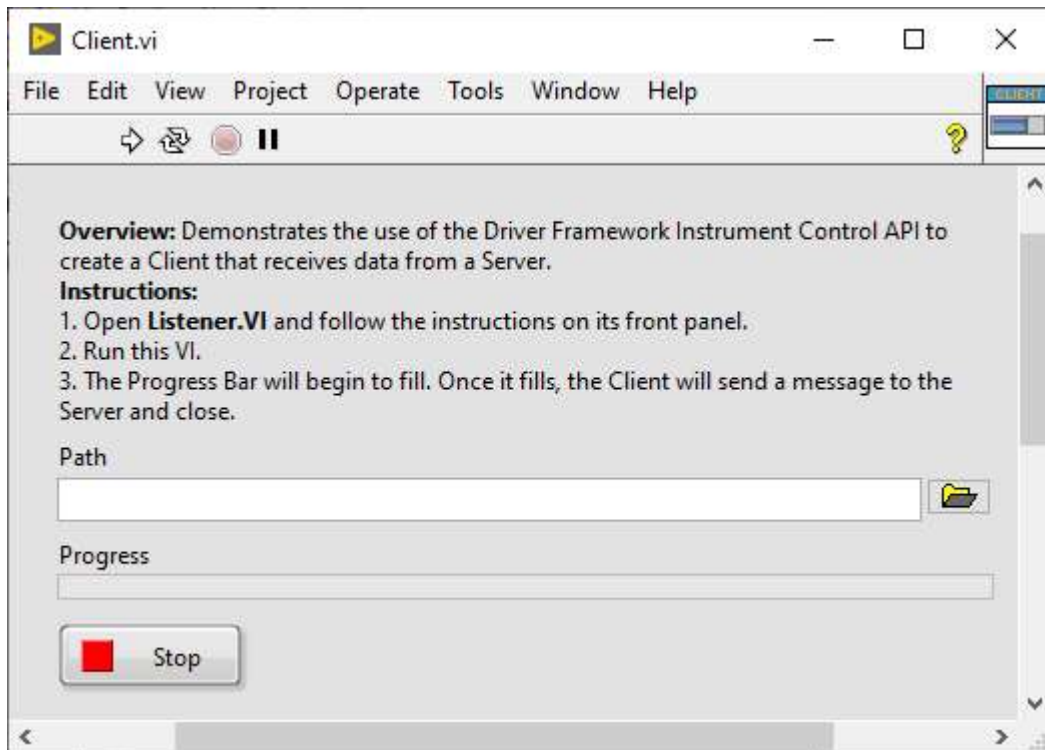


Figure 15: Client Demo Front Panel

The Client establishes a connection with the Listener and then reads data from it. Once the Client counts to 100, it stops and sends a message to the Listener, telling it to stop.

5.1.2.1. Initialize

Like the Listener, the Client will need to initialize its instrument control. Since the names are used to keep track of the instrument control in the Console, it must have a unique name.

5.1.2.2. Load

Now the Client will have to load its instrument control from file.

5.1.2.3. *Open*

Once the instrument control is successfully loaded, it needs to establish a connection with the Listener. It is important that the Listener is already running at this point. Once the connection is established, the Client will continually read data from the Listener.

5.1.2.4. *Read Bytes*

The Client expects data from the Listener in two packets. The first packet is a single character that tell the Client how many characters will be in the following packet. If this number is zero, it doesn't look for a second packet. If it is a non-zero value, then it reads that many bytes from the Listener. This data is expected to be a number, so it is converted and checked. If the number from the Listener is 100 or greater, the Client beings to shut down.

5.1.2.5. *Write to Instrument*

In this program, the Client needs to tell the Listener that it is shutting down. It does this by writing a 'Q' to the Listener. The Listener will interpret this as a command to shut down.

5.1.2.6. *Close*

Once the Listener has been told to shut down, the last thing to do is close the connection. This also informs the Console that the Client's instrument control has been closed.

5.1.3. Configuration

This section outlines how to use the CMC Configuration Manager to create a configuration file. For more details on the CMC Configuration Manager, see Section 4.1.

5.1.3.1. Listener Configuration

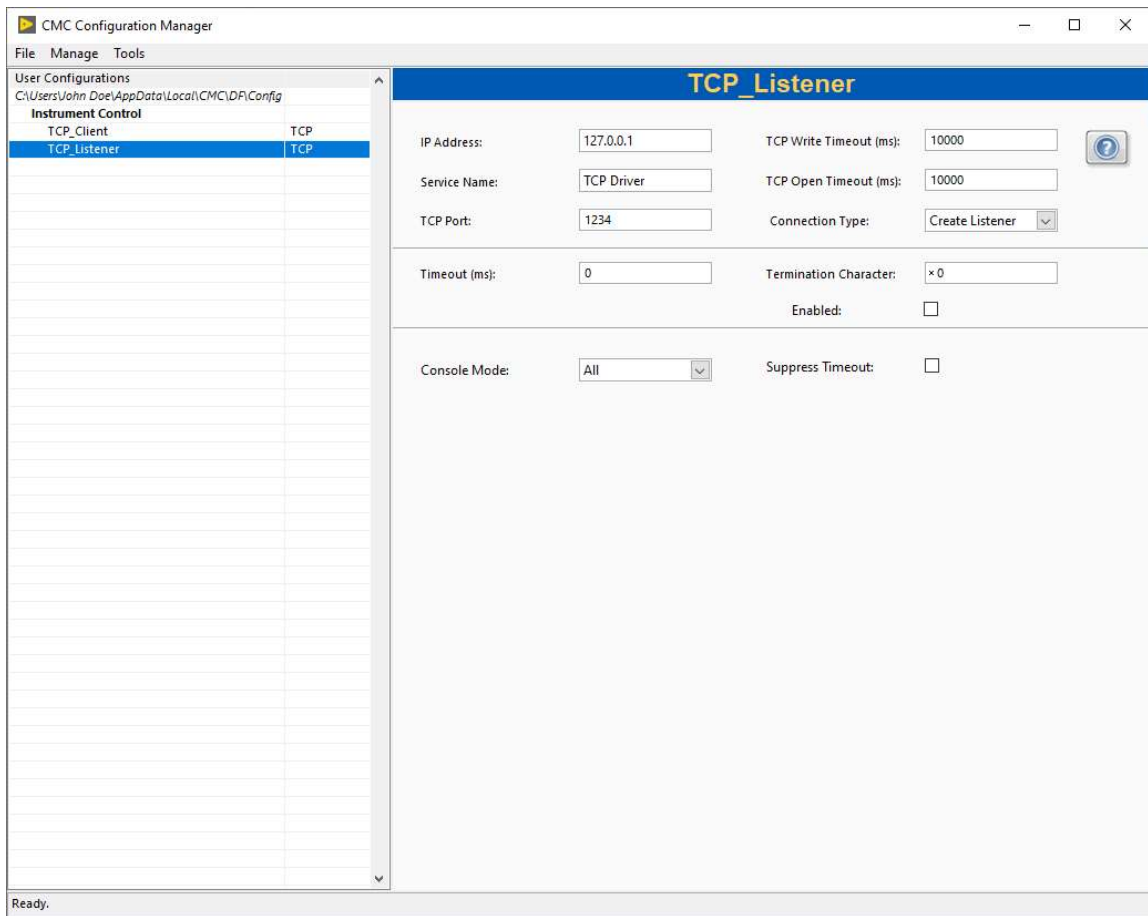


Figure 16: Demo Listener Configuration

1. In LabVIEW, go to Tools >> CMC Tools >> CMC Configuration Manager.
2. In the CMC Configuration Manager, go to the File menu and select Create New. This will bring up the Create New Configuration dialog.
3. In the Create New dialog, give the configuration a unique name. For this tutorial, it will be assumed that it is named "My Test Listener".
4. Select TCP (Instrument >> Instrument Control >> TCP) as the instrument type.
5. Click OK.
6. In the file dialog that pops up, click Ok.
7. Set the IP Address to 127.0.0.1.
8. Set the Connection Type to be "Create Listener".
9. Set the Port to 1234. If this port is already used, enter an unused port number. Ensure that any local firewall settings do not block LabVIEW from using this port.
10. Select File >> Save, or press Ctrl+S to save the changes to the .instconfig.

5.1.3.2. Client Configuration

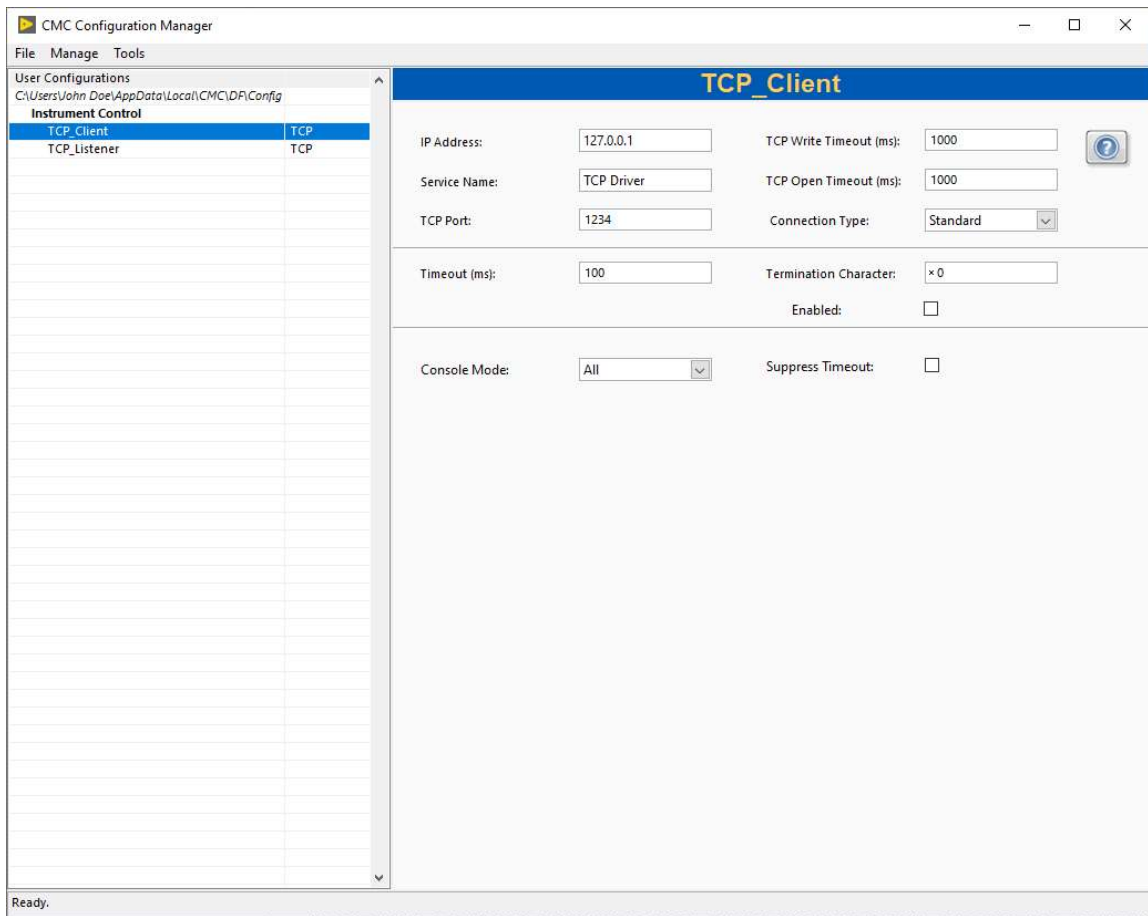


Figure 17: Demo Client Configuration

1. In the CMC Configuration Manager, go to the File menu and select Create New. This will bring up the Create New Configuration dialog.
2. In the Create New Configuration dialog, give the configuration a unique name. For this tutorial, it will be assumed that it is named "My Test Client".
3. Select TCP (Instrument >> Instrument Control >> TCP) as the instrument type.
4. Click OK.
5. In the file dialog that pops up, click Ok.
6. Set the IP Address to 127.0.0.1.
7. Set the Connection Type to be "Standard Connection".
8. Set the Port to the port number chosen for the Listener Configuration.
9. Select File >> Save, or press Ctrl+S to save the changes to the .instconfig.

5.1.4. Run Demo

In order to test that the newly created instconfig files work correctly, change the paths on the demo front panels to point to the new instconfig files. These can be found in the user's application data directory. On Windows 10, this is found at C:\Users\

demo may be run. If both files are configured correctly, the Client will start filling its progress bar. Once the bar is full, both VIs will terminate.

5.2. Console Tutorial

This tutorial will provide an overview of Console. The Console provides an interface where the communication with one or more instruments can be observed. This tutorial will make use of the demo VIs. Since these VIs are discussed in more detail in the previous tutorial (see Section 5.1), it is recommended to read through that before starting this tutorial.

The Console can be launched by the Initialize VI from the Instrument Control Palette or by the Launch Console VI from the Console Palette. For more details on the various components that compose the Console, see Section 4.2.

5.2.1. Viewing Console Logs

One of the uses of the Console is to provide the user with a window into the communication of the instrument to monitor the messages that are sent and received. This can be helpful for debugging unexpected instrument behavior.

Open the Console using the Launch Console VI from the CMC >> Console Palette. When the Console is initially launched, there won't be any instruments registered with it, so the Instrument Selector will be empty, as well as the Console Log.

In order to populate the Console's Instrument Selector and Console Log, run the TCP Listener Demo VI with the TCP Client Demo VI. Once they begin execution, the Console will automatically select the Listener in the Instrument Selector, since the Console didn't have anything selected and the Listener will initialize first. While they execute, their Console logs will update as they communicate with each other. These can be switched between by selecting the desired instrument with the Instrument Selector.

Once the TCP Listener and Client Demo VIs finish executing, the Console will remain open with their respective logs still available for viewing. When the Console is closed, the logs will be saved to file.

5.2.2. Injection

Another use of the Console is to provide an interface that enables the user to send messages to the instrument during execution.

For example, the TCP Listener Demo VI can be prematurely shut down using console injection by the following steps:

1. Start the TCP Listener Demo VI and TCP Client Demo VI.
2. With the Client selected in the Console, enter a 'Q' into the Injector Message.
3. Click the Write button or press enter. This inserts a 'Q' that is sent to the instrument that the Client is controlling, in this case, the Listener. The Listener will read this message and shutdown. However, since the Client hasn't finished counting, it will continue to look for messages from the Listener.
4. Click the stop button to stop the Client.

Similarly, the Client can also be shut down early by console injection.

1. Start the TCP Listener Demo VI and TCP Client Demo VI.
2. Select the Listener in the Instrument Selector of the Console.
3. Type '3100' into the Injector Message.
4. Click the Write button or press enter.

The Client will read this in as two separate messages. The first message, the 3, will inform it of the length of the second message, the 100. It will read in the 100 and since that is greater than 99, it will run its normal shutdown procedure, which will also shut down the Listener.

5.3. TCP Tutorial

This tutorial will walk the user through creating a simple client/listener pair along with the required configuration files. The client (UI) will be a simple UI for the Listener (Processor), which will do all the processing.

This tutorial will use common controls, indicators and programming functions and assumes some amount of familiarity with the LabVIEW programming environment. It will make use of the Instrument Control API, which can be found in the block diagram palette set under CMC >> Instrument Control, see Section 4.3.1 for more details.

5.3.1. Creating UI

5.3.1.1. Configuration


CMCTutorialUI				
IP Address:	<input type="text" value="127.0.0.1"/>	TCP Write Timeout (ms):	<input type="text" value="2000"/>	
Service Name:	<input type="text"/>	TCP Open Timeout (ms):	<input type="text" value="20000"/>	
TCP Port:	<input type="text" value="1234"/>	Connection Type:	<input type="text" value="Standard"/>	
Timeout (ms):	<input type="text" value="1000"/>	Termination Character:	<input type="text" value="xA"/>	
		Enabled:	<input type="checkbox"/>	
Console Mode:	<input type="text" value="None"/>	Suppress Timeout:	<input type="checkbox"/>	

Figure 18: UI Configuration

The UI will need an .instconfig in order to actually know what type of Instrument Control it should be. In order to create it, use the following steps:

1. Open the CMC Configuration Manager. (Tools >> CMC Tools >> CMC Configuration Manager...)
2. In the File Menu, select Create New.
3. For the Name, enter CMCTutorialUI. If this turns red, click Cancel and skip to step 7. An instconfig with this name already exists.

4. Select TCP as the instrument type.
5. Click OK.
6. Click OK in the file dialog that pops up.
7. In the CMC Configuration Manager, make sure that the configuration named CMCTutorialUI is selected.
8. Set the IP Address to 127.0.0.1. This is the local host IP Address. Since there isn't an external instrument to connect to, using the local host is a safe option. For a newly created .instconfig, the IP Address will default to 127.0.0.1.
9. Decide if a Service Name would be useful. For this tutorial, it will not, change it to be blank.
10. Select the Connection Type. Since the UI will be the client, set the Connection Type to Standard Connection.
11. Pick an unused TCP port for the communication to be transmitted over. Assuming 1234 is not used for anything, use that.
12. Decide how long the TCP connection will wait for a Write action to complete. For this tutorial, the default value of 2000 milliseconds will be fine.
13. Set the TCP Open Timeout to be long enough for the UI to connect to the Processor. For the UI, 20000 milliseconds will work.
14. Set the Console to None. The Console will not be used in this tutorial, so it doesn't need to be enabled.
15. Set the Timeout to 1000 milliseconds. This will be important so that our query doesn't timeout before the Processor can finish calculating and returning its response.
16. Uncheck Enabled. This will remove the termination character from being added to the transmission.
17. Save the Configuration and exit.

5.3.1.2. Front Panel

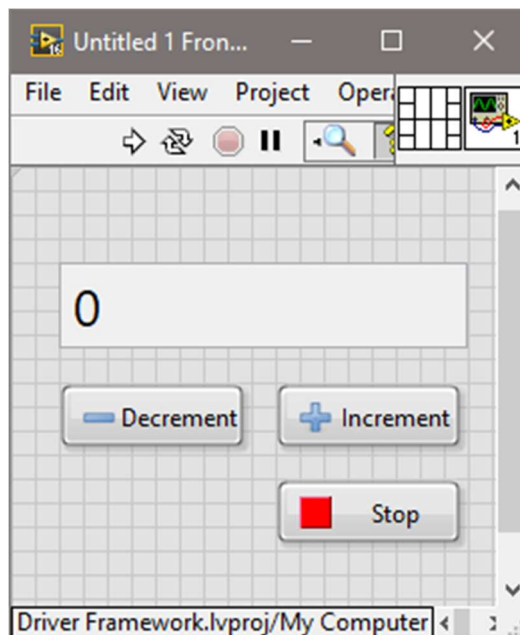


Figure 19: UI Front Panel

The front panel will consist of the following controls and indicators:

- Decrement Button.
 - Silver >> Boolean >> Buttons >> Remove Button (Silver).
 - Change the Label to “Decrement Button”.
- Increment Button.
 - Silver >> Boolean >> Buttons >> Add Button (Silver).
 - Change the Label to “Increment Button”.
- Stop Button.
 - Silver >> Boolean >> Buttons >> Stop Button (Silver).
 - Change the Label to “Stop Button”.
- Numeric Indicator.
 - System >> Numeric >> System Numeric.
 - Change the Label to “Value”.
 - Right-click >> Change to indicator.
 - Right-click >> Representation >> I32.

5.3.1.3. Block Diagram

The block diagram will start off with setting up the instrument control. Then it will move to the main loop, which will handle button press events. Finally, it will close the instrument control.

5.3.1.3.1. Instrument Control Setup

Initialize the Instrument Control, as shown in Figure 20.

- Place an Initialize VI from the CMC >> Instrument Control Palette onto the Block Diagram.
- Place a String Constant from the Programming >> String Palette onto the Block Diagram.
- Enter “UI” into the String Constant.
- Wire the String Constant to the **Instrument Name** terminal of the Instrument Control Initialize VI.



Figure 20: UI Instrument Control Initialization

Load the instconfig into the Instrument Control, as shown in Figure 21.

- Place a Get System Directory VI from the Programming >> File I/O >> File Constants Palette onto the Block Diagram.
- Right click the **system directory type** terminal and select Create Constant.
- In the newly created enum, select User Application Data.
- Place a Build Path function from the Programming >> File I/O Palette onto the Block Diagram.
- Wire the **system directory** terminal from the Get System Directory VI to the **base path** terminal of the Build Path Function.

- Place a String Constant from the Programming >> String Palette onto the Block Diagram.
- Enter “CMC\DF\Config\CMCTutorialUI.instconfig” into the String Constant.
- Wire the String Constant to the **name or relative path** terminal of the Build Path function.
- Place a Load VI from the CMC >> Instrument Control Palette onto the Block Diagram.
- Wire the **Instrument** output terminal from the Instrument Control Initialize VI to the **Instrument** input terminal of the Instrument Control Load VI.
- Wire the **error out** terminal from the Instrument Control Initialize VI to the **error in (no error)** terminal of the Instrument Control Load VI.
- Wire the **appended path** terminal from the Build Path function to the **Configuration File** terminal of the Instrument Control Load VI.

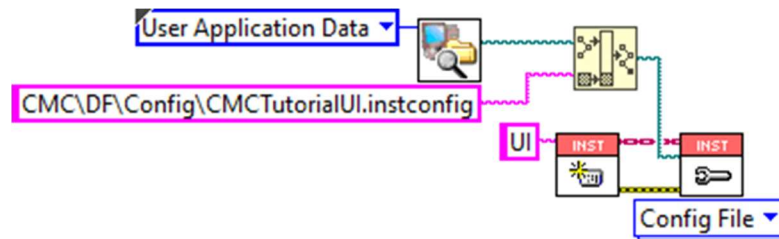


Figure 21: UI Instrument Control Load Instrument

Open the communications shown in Figure 22.

- Place an Open VI from the CMC >> Instrument Control Palette onto the block Diagram.
- Wire the **Instrument** output terminal from the Instrument Control Load VI to the **Instrument** input terminal of the Instrument Control Open VI.
- Wire the **error out** terminal of the Instrument Control Load VI to the **error in (no error)** terminal of the Instrument Control Open VI.

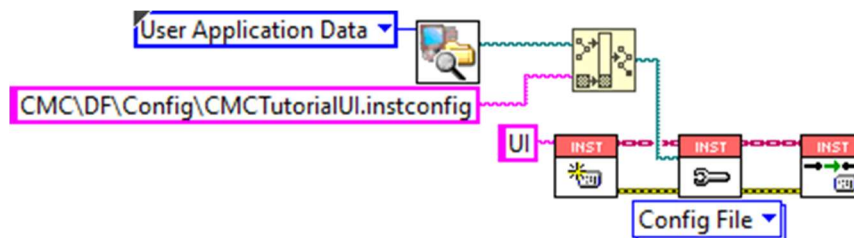


Figure 22: UI Instrument Control Open

5.3.1.3.2. Main Process

Now that the Instrument Control has been created, configured and opened, the communication process needs to be built up.

Create the basic architecture, as shown in Figure 23.

- Place a Case Structure from the Programming >> Structures Palette onto the Block Diagram.

- Wire the **Instrument** output terminal from the Instrument Control Open VI into the Case Structure.
- Wire the **error out** terminal from the Instrument Control Open VI to the Case Selector of the Case Structure.
- Place a While Loop from the Programming >> Structures Palette inside the Case Structure.
- Wire the **Instrument** tunnel from the Case Structure into the While Loop.
- Wire the **Error** tunnel from the Case Structure into the While Loop.
- Place an Event Structure from the Programming >> Structures Palette inside the While Loop.

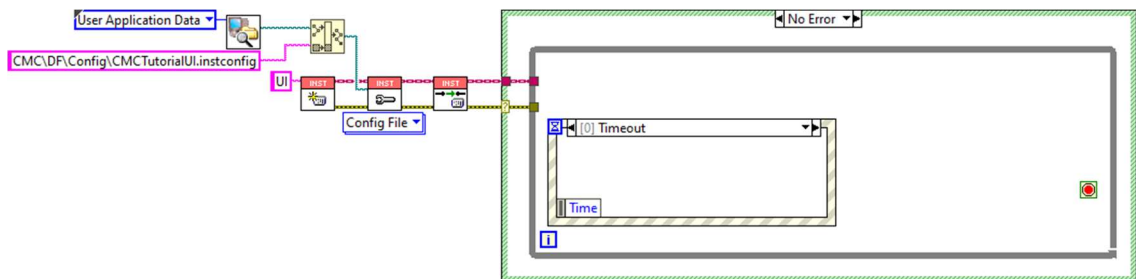


Figure 23: UI Main Process Loop

Handle the Error Case, as shown in Figure 24.

- Wire the **Instrument** tunnel through the “Error” case of the Case Structure.
- Wire the **Error** tunnel through the “Error” case of the Case Structure.

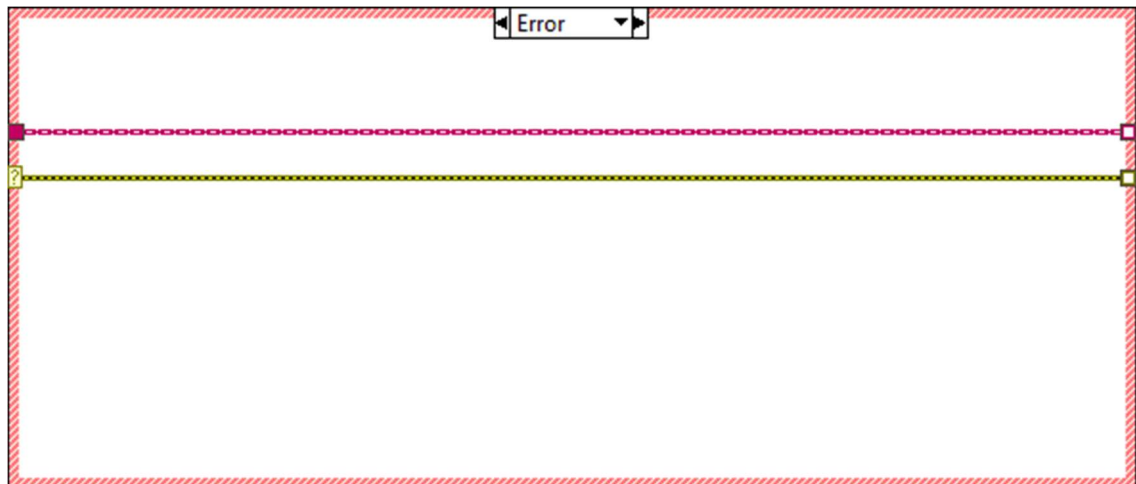


Figure 24: UI Main Process Error

Create the base process and fill in the Timeout Event of the Event Structure, as shown in Figure 25.

- Place a String Constant from the Programming >> String Palette inside the Timeout Event of the Event Structure.
- Enter “?” into the String Constant.
- Wire the String Constant to the Event Structure as an output tunnel.

- Place a False Constant from the Programming >> Boolean Palette inside the Timeout Event of the Event Structure.
- Wire the False Constant to the Event Structure as an output tunnel.
- Place a Read/Write Instrument VI from the CMC >> Instrument Control Palette inside the While Loop.
- Set the Instrument Control Read/Write Instrument polymorphic VI to use the “Query (Data)” instance.
- Wire the **Instrument** tunnel from the While Loop to the **Instrument** input terminal of the Instrument Control Query (Data) VI.
- Wire the **Instrument** output terminal from the Instrument Control Query (Data) VI to the **Instrument** output tunnel of the Case Structure.
- Wire the **Error** tunnel from the While Loop to the **error in (no error)** terminal of the Instrument Control Query (Data) VI.
- Wire the **error out** terminal of the Instrument Control Query (Data) VI to the **Error** output tunnel of the Case Structure.
- Wire the String output from the Event Structure to the **Data** input terminal of the Instrument Control Query (Data) VI.
- Place a String Constant from the Programming >> String Palette inside the While Loop.
- Enter “A” into the String Constant.
- Wire the String Constant to the **Desired Data** terminal of the Instrument Control Query (Data) VI.
- Place a Decimal String to Number from the Programming >> String >> String/Number Conversion Palette inside the While Loop.
- Wire the **Data** output terminal from the Instrument Control Query (Data) VI to the **String** terminal of the Decimal String to Number function.
- Move the terminal of the Value Indicator inside the While Loop.
- Wire the **number** terminal from the Decimal String to Number to the terminal of the Value Indicator.
- Wire the Boolean from the Event Structure to the Conditional Terminal of the While Loop.

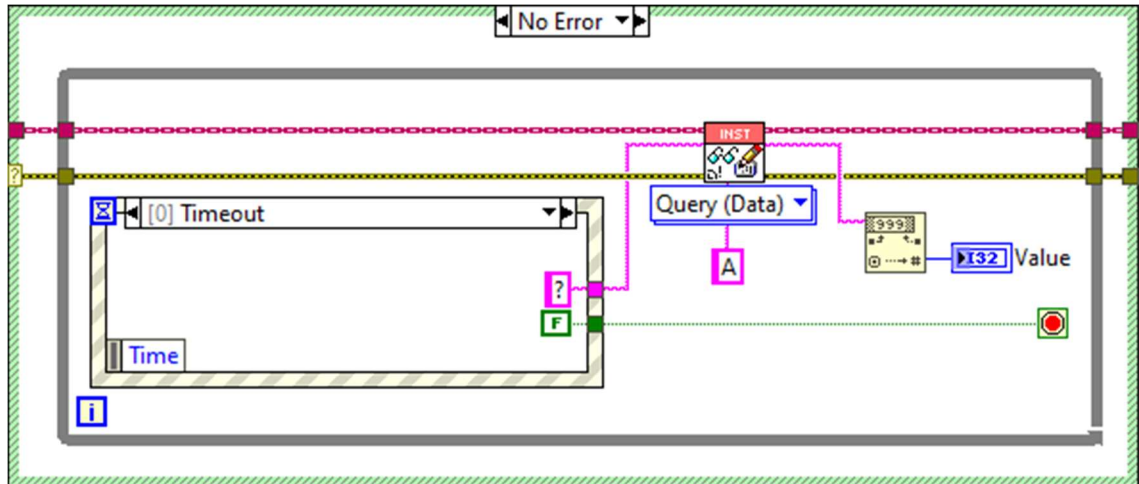


Figure 25: UI Main Process Timeout

Configure the Increment Message, as shown in Figure 26.

- Right-click the Event Structure and select “Add Event Case”.
- Select “Increment Button” as the Event Source.
- Select “Value Change” as the Event.
- Click “OK”.
- Place a String Constant from the Programming >> String Palette inside the “Increment Button”: Value Change Event Case.
- Enter “+” into the String Constant.
- Wire the String Constant to the String tunnel of the Event Structure.
- Place a False Constant from the Programming >> Boolean Palette inside the “Increment Button”: Value Change Event Case.
- Wire the False Constant to the Boolean tunnel of the Event Structure.

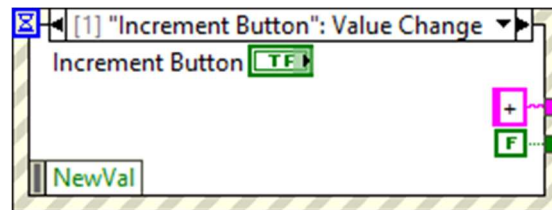


Figure 26: UI Main Process Add

Configure the Decrement Message, as shown in Figure 27.

- Right-click the Event Structure and select “Add Event Case”.
- Select “Decrement Button” as the Event Source.
- Select “Value Change” as the Event.
- Click “OK”.
- Place a String Constant from the Programming >> String Palette inside the “Decrement Button”: Value Change Event Case.
- Enter “-” into the String Constant.
- Wire the String Constant to the String tunnel of the Event Structure.

- Place a False Constant from the Programming >> Boolean Palette inside the “Decrement Button”: Value Change Event Case.
- Wire the False Constant to the Boolean tunnel of the Event Structure.

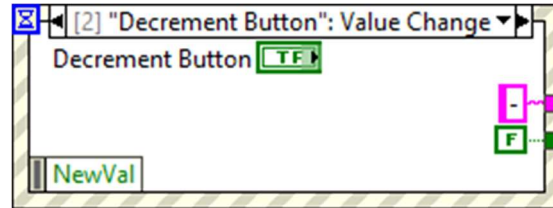


Figure 27: UI Main Process Subtract

Configure the Stop Message, as shown in Figure 28.

- Right-click the Event Structure and select “Add Event Case”.
- Select “Stop Button” as the Event Source.
- Select “Value Change” as the Event.
- Click “OK”.
- Place a String Constant from the Programming >> String Palette inside the “Stop Button”: Value Change Event Case.
- Enter “X” into the String Constant.
- Wire the String Constant to the String tunnel of the Event Structure.
- Place a True Constant from the Programming >> Boolean Palette inside the “Stop Button”: Value Change Event Case.
- Wire the True Constant to the Boolean tunnel of the Event Structure.

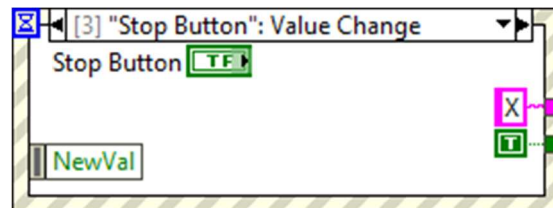


Figure 28: UI Main Process Stop Event

The last event needed is for the Stop Button. When this button is pressed, the UI will send an ‘X’ to the Processor. This case will also have a TRUE wired to the While Loop terminal.

5.3.1.3.3. Close

Close the Instrument Control, as shown in Figure 29.

- Place a Close VI from the CMC >> Instrument Control Palette onto the Block Diagram.
- Wire the **Instrument** output tunnel from the Case Structure to the **Instrument** input terminal of the Instrument Control Close VI.
- Wire the **Error** output tunnel from the Case Structure to the **error in (no error)** terminal of the Instrument Control Close VI.

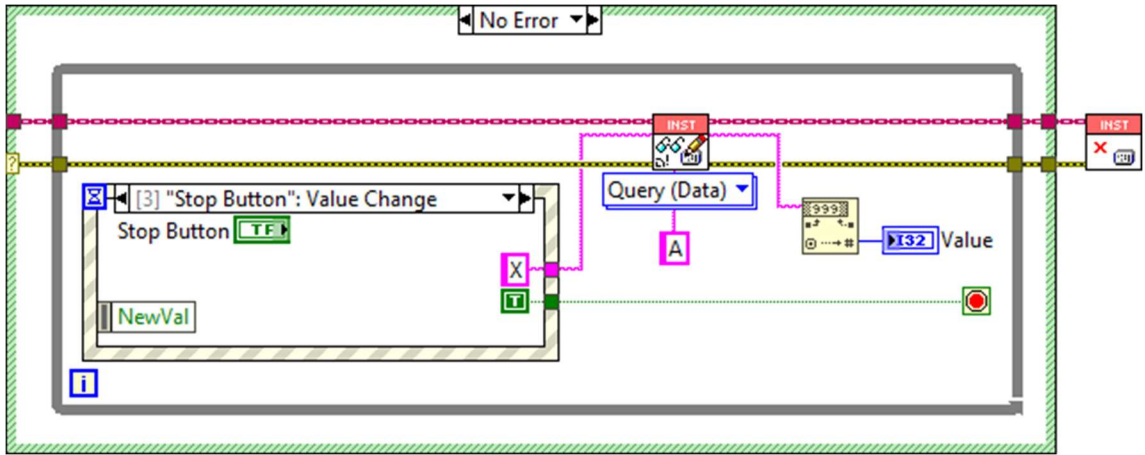
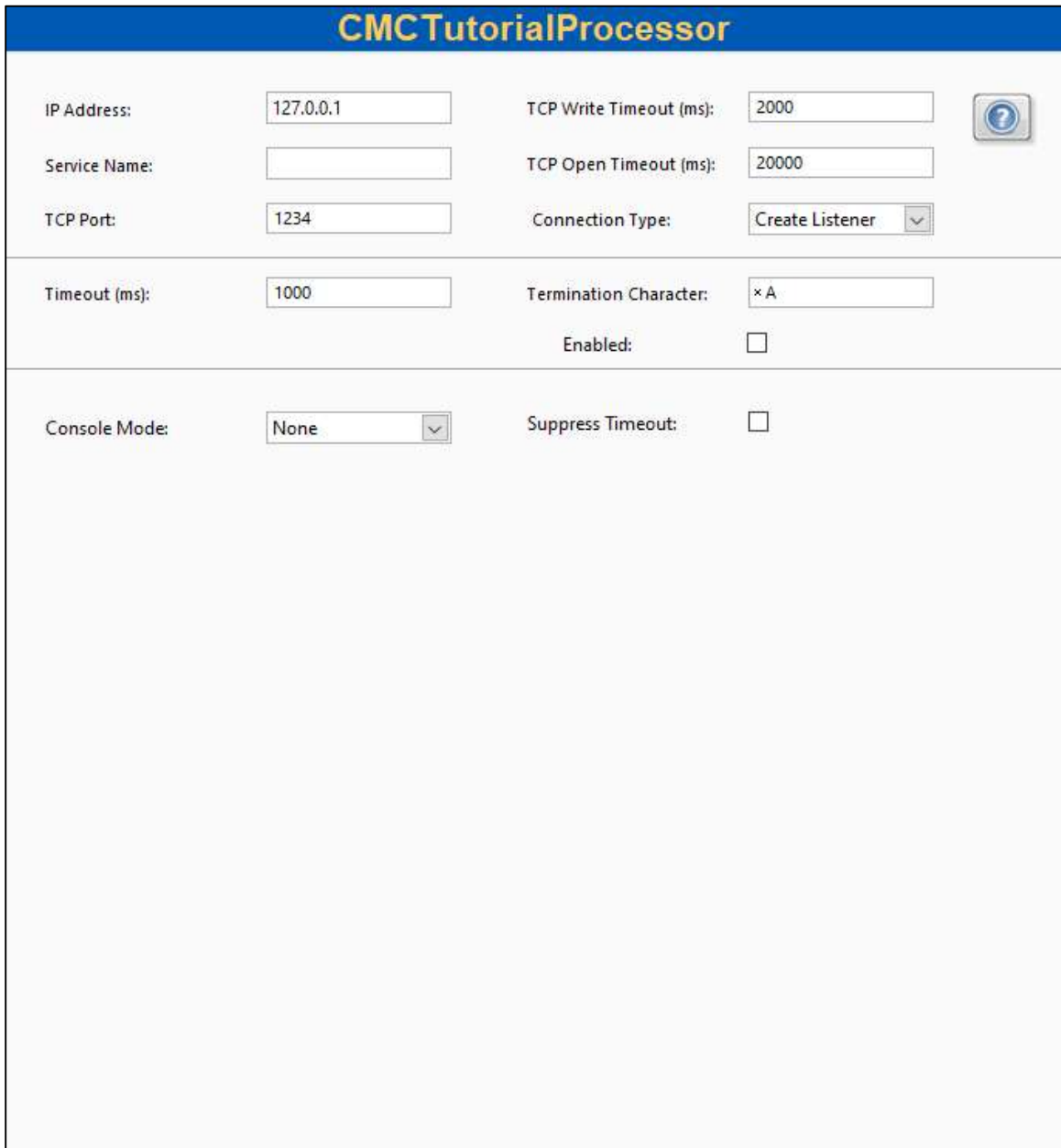


Figure 29: UI Close

The Main process is now complete.

5.3.2. Creating Processor

5.3.2.1. Configuration



The image shows a configuration window titled "CMCTutorialProcessor". It contains several input fields and checkboxes for configuring a processor. The fields are arranged in a grid-like fashion. A help icon (a question mark in a square) is located in the top right corner of the dialog.


CMCTutorialProcessor				
IP Address:	<input type="text" value="127.0.0.1"/>	TCP Write Timeout (ms):	<input type="text" value="2000"/>	
Service Name:	<input type="text"/>	TCP Open Timeout (ms):	<input type="text" value="20000"/>	
TCP Port:	<input type="text" value="1234"/>	Connection Type:	<input type="text" value="Create Listener"/>	
Timeout (ms):	<input type="text" value="1000"/>	Termination Character:	<input type="text" value="xA"/>	
		Enabled:	<input type="checkbox"/>	
Console Mode:	<input type="text" value="None"/>	Suppress Timeout:	<input type="checkbox"/>	

Figure 30: Processor Configuration

The Processor will need an .instconfig in order to actually know what type of instrument control it should be. In order to create it, use the following steps:

1. Open the CMC Configuration Manager. (Tools >> CMC Tools >> CMC Configuration Manager...)
2. In the File Menu, select Create New.
3. For the Name, enter CMCTutorialProcessor. If this turns red, click Cancel and skip to step 7. An instconfig with this name already exists.
4. Select TCP as the instrument type.

5. Click OK.
6. Click OK in the file dialog that pops up.
7. In the CMC Configuration Manager, make sure that the configuration named CMCTutorialProcessor is selected.
8. Set the IP Address to 127.0.0.1. This is the local host IP Address. Since there isn't an external instrument to connect to, using the local host is a safe option. For a newly created .instconfig, the IP Address will default to 127.0.0.1. In order for the UI to communicate with the Processor, both must be configured to use the same IP Address.
9. Decide if a Service Name would be useful. For this tutorial, it will not, change it to be blank.
10. Select the Connection Type. Since the Processor will be the Listener, set the Connection Type to Create Listener.
11. Pick the same TCP port that was used for the UI's configuration for the communication to be transmitted over.
12. Decide how long the TCP connection will wait for a Write action to complete. For this tutorial, the default value of 2000 milliseconds will be fine.
13. Set the TCP Open Timeout to be long enough for the UI to connect to the Processor. For the UI, 20000 milliseconds will work.
14. Set the Console to None. The Console will not be used in this tutorial, so it doesn't need to be enabled.
15. Set the Timeout to 1000 milliseconds. This is not particularly important since the Processor will wait forever for a message from the UI. Once it gets the message, it will perform some action and return the updated Value to the UI.
16. Uncheck Enabled. This will remove the termination character from being added to the transmission.
17. Save the Configuration and exit.

5.3.2.2. *Front Panel*

The Processor doesn't have anything on its front panel.

5.3.2.3. *Block Diagram*

The Block Diagram will setup the Instrument Control, manage communication with the UI and close the Instrument Control once the Processor is no longer needed.

5.3.2.3.1. *Instrument Control Setup*

Initialize the Instrument Control, as shown in Figure 31.

- Place an Initialize VI from the CMC >> Instrument Control Palette onto the Block Diagram.
- Place a String Constant from the Programming >> String Palette onto the Block Diagram.
- Enter "Processor" into the String Constant.
- Wire the String Constant to the **Instrument Name** terminal of the Instrument Control Initialize VI.

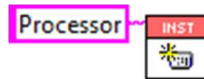


Figure 31: Processor Instrument Control Initialize

Load the instconfig into the Instrument Control, as shown in Figure 32.

- Place a Get System Directory VI from the Programming >> File I/O >> File Constants Palette onto the Block Diagram.
- Right click the **system directory type** terminal and select Create Constant.
- In the newly created enum, select User Application Data.
- Place a Build Path function from the Programming >> File I/O Palette onto the Block Diagram.
- Wire the **system directory** terminal from the Get System Directory VI to the **base path** terminal of the Build Path Function.
- Place a String Constant from the Programming >> String Palette onto the Block Diagram.
- Enter “CMC\DF\Config\CMCTutorialProcessor.instconfig” into the String Constant.
- Wire the String Constant to the **name or relative path** terminal of the Build Path function.
- Place a Load VI from the CMC >> Instrument Control Palette onto the Block Diagram.
- Wire the **Instrument** output terminal from the Instrument Control Initialize VI to the **Instrument** input terminal of the Instrument Control Load VI.
- Wire the **error out** terminal from the Instrument Control Initialize VI to the **error in (no error)** terminal of the Instrument Control Load VI.
- Wire the **appended path** terminal from the Build Path function to the **Configuration File** terminal of the Instrument Control Load VI.

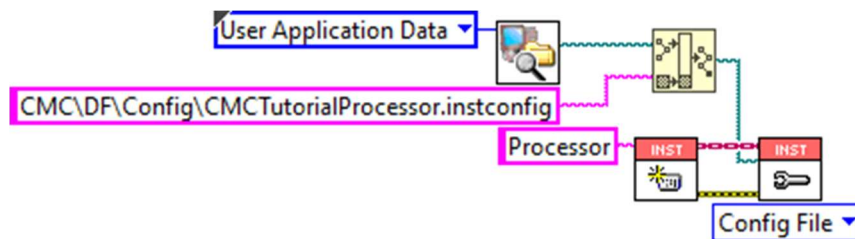


Figure 32: Processor Instrument Control Load

Open the communications, as shown in Figure 33.

- Place an Open VI from the CMC >> Instrument Control Palette onto the block Diagram.
- Wire the **Instrument** output terminal from the Instrument Control Load VI to the **Instrument** input terminal of the Instrument Control Open VI.
- Wire the **error out** terminal of the Instrument Control Load VI to the **error in (no error)** terminal of the Instrument Control Open VI.

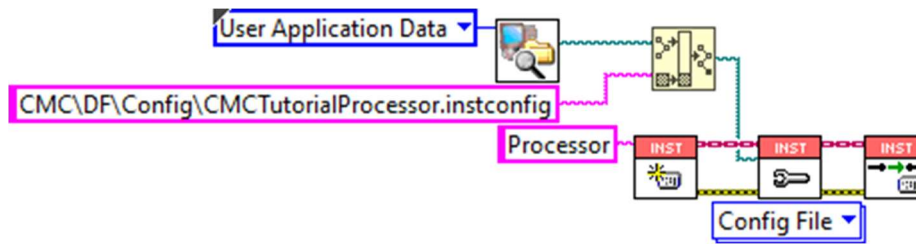


Figure 33: Processor Instrument Control Open

5.3.2.3.2. Main process

Now that the Instrument Control has been created, configured and opened, the process to handle the messages from the UI needs to be made.

Create the basic architecture, as shown in Figure 34.

- Place a Case Structure from the Programming >> Structures Palette onto the Block Diagram.
- Wire the **Instrument** output terminal from the Instrument Control Open VI into the Case Structure.
- Wire the **error out** terminal from the Instrument Control Open VI to the Case Selector of the Case Structure.
- Place a While Loop from the Programming >> Structures Palette inside the Case Structure.
- Wire the **Instrument** tunnel from the Case Structure into the While Loop.
- Wire the **Error** tunnel from the Case Structure into the While Loop.

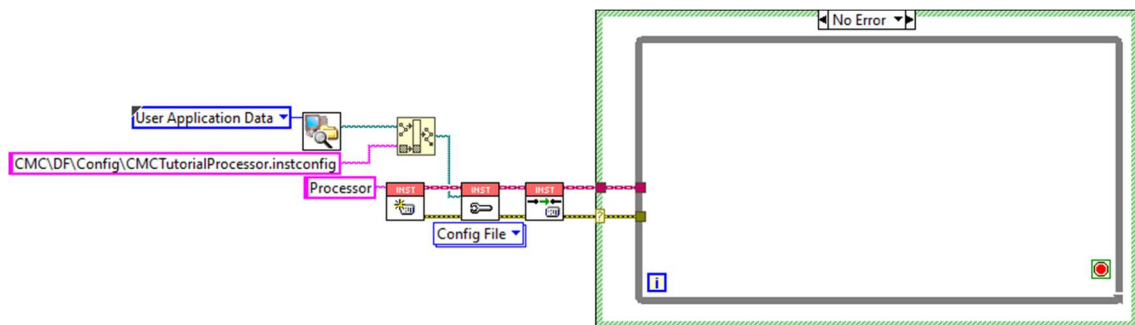


Figure 34: Processor Main Loop

Handle the Error Case, as shown in Figure 35.

- Wire the **Instrument** tunnel through the "Error" case of the Case Structure.
- Wire the **Error** tunnel through the "Error" case of the Case Structure.

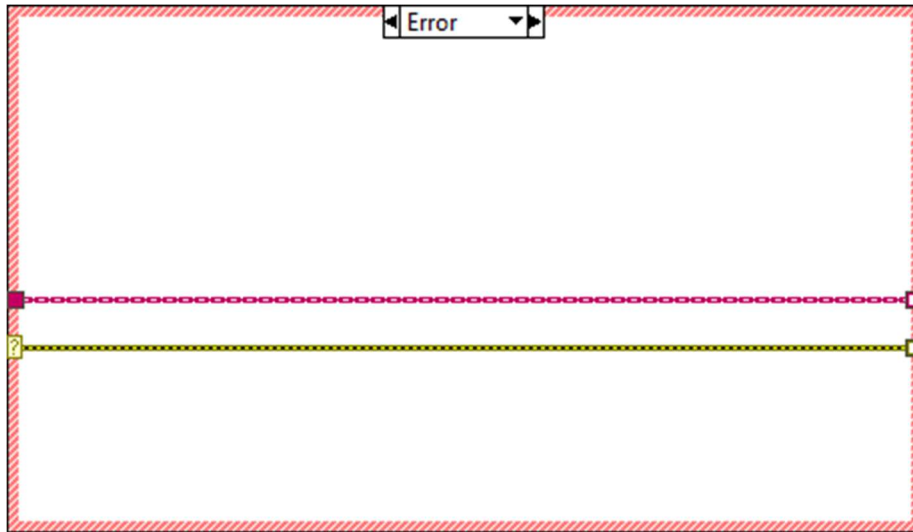


Figure 35: Processor Main Error

Add code so that the Processor will read messages from the UI and respond, as shown in Figure 36.

- Place a Read/Write Instrument VI from the CMC >> Instrument Control Palette inside the While Loop.
- Set the Instrument Control Read/Write Instrument polymorphic VI to use the “Read (Bytes)” instance.
- Wire the **Instrument** tunnel of the While Loop to the **Instrument** input terminal of the Instrument Control Read (Bytes) VI.
- Wire the **Error** tunnel of the While Loop to the **error in (no error)** terminal of the Instrument Control Read (Bytes) VI.
- Place an Equal? function from the Programming >> Comparison Palette inside the While Loop.
- Wire the **Data** terminal of the Instrument Control Read (Bytes) VI to the **x** terminal of the Equal? function.
- Place a String Constant from the Programming >> String Palette inside the While Loop.
- Enter “X” into the String Constant.
- Wire the String Constant to the **y** terminal of the Equal? function.
- Wire the **x =y?** terminal from the Equal? function to the Conditional Terminal of the While Loop.
- Place a Format Into String function from the Programming >> String Palette inside the While Loop.
- Wire the **error out** terminal of the Instrument Control Read (Bytes) VI to the **error in** terminal of the Format Into String function.
- Place a String Constant from the Programming >> String Palette inside the While Loop.
- Enter “%dA” into the String Constant.
- Wire the String Constant to the **format string** terminal of the Format Into String function.

- Place a Read/Write Instrument VI from the CMC >> Instrument Control Palette inside the While Loop.
- Set the Instrument Control Read/Write Instrument polymorphic VI to use the “Write” instance.
- Wire the **Instrument** output terminal from the Instrument Control Read (Bytes) VI to the **Instrument** input terminal of the Instrument Control Write VI.
- Wire the **resulting string** terminal from the Format Into String function to the **Data** terminal of the Instrument Control Write VI.
- Wire the **error out** terminal from the Format Into String function to the **error in (no error)** terminal of the Instrument Control Write VI.
- Wire the **Instrument** output terminal from the Instrument Control Write VI to the **Instrument** tunnel of the Case Structure.
- Wire the **Instrument** output terminal from the Instrument Control Write VI to the **Instrument** tunnel of the Case Structure.

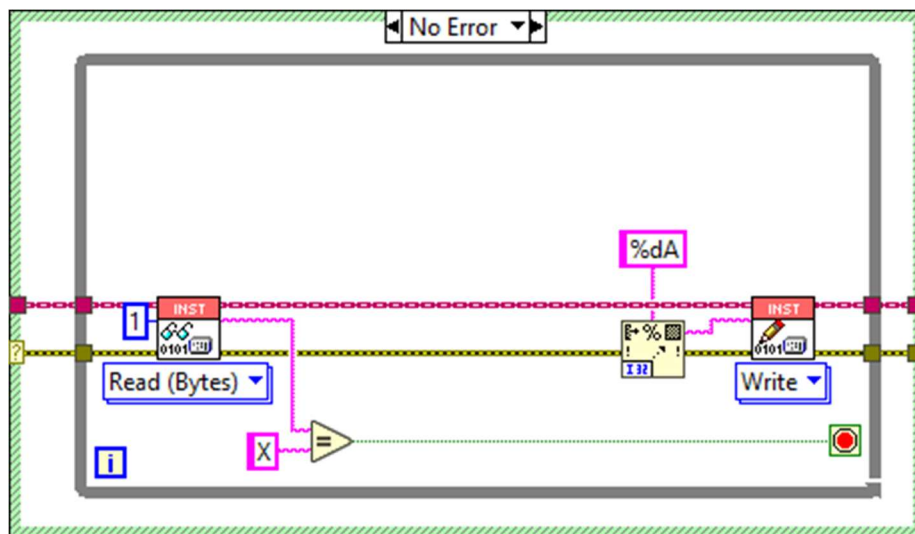


Figure 36: Processor Main Loop Base

Add a variable and message processing, as shown in Figure 37.

- Right-click on the While Loop and select “Add Shift Register”.
- Place a Numeric Constant from the Programming >> Numeric Palette inside the Case Structure, but outside the While Loop.
- Wire the Numeric Constant to the Shift Register.
- Place a Case Structure from the Programming >> Structures Palette inside the While Loop.
- Wire the **Data** terminal from the Instrument Control Read (Bytes) VI to the Case Selector of the Case Structure.
- Select the default case of the Case Structure.
- Replace “False” with “?” in the Selector Label of the Case Structure.
- Wire the Numeric Constant from the Shift Register through the default case of the Case Structure and into the other side of the Shift Register.

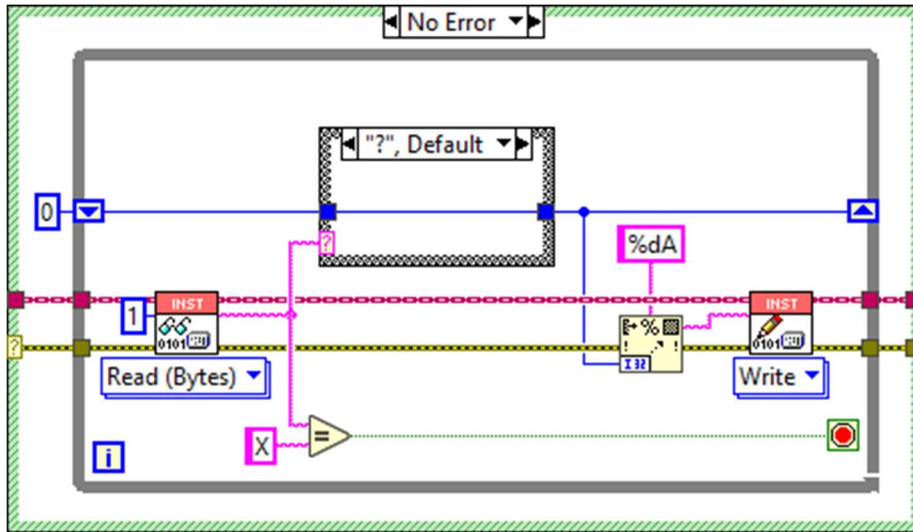


Figure 37: Processor Main Loop Read Write

Handle the decrement message, as shown in Figure 38.

- Select the “True” case of the Case Structure.
- Replace “True” with “-”.
- Place a Decrement function from the Programming >> Numeric Palette inside the Case Structure.
- Wire the Numeric Constant input tunnel to the x terminal of the Decrement function.
- Wire the $x-1$ terminal of the Decrement function to the Numeric Constant output tunnel.

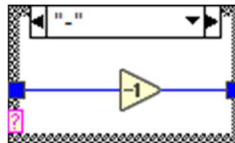


Figure 38: Processor Main Loop Decrement Message

Handle the Increment message, as shown in Figure 39.

- Right click the Case Structure and select “Add Case After”.
- Enter “+” into the Case Selector.
- Place an Increment function from the Programming >> Numeric Palette inside the Case Structure.
- Wire the Numeric Constant input tunnel to the x terminal of the Increment function.
- Wire the $x+1$ terminal of the Increment function to the Numeric Constant output tunnel.

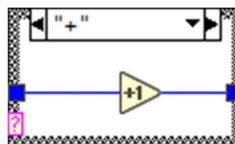


Figure 39: Processor Main Loop Increment Message

Handle the Close message, as shown in Figure 40.

- Right click the Case Structure and select “Add Case After”.
- Enter “X” into the Case Selector.
- Wire the Numeric Constant input tunnel to the Numeric Constant output tunnel.

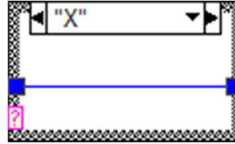


Figure 40: Processor Main Loop Close Message

The Processor’s message handling is now complete.

5.3.2.3.3. Close

With all the functionality of the Processor completed, once it finishes executing, the Instrument Control will need to be close, as shown in Figure 41.

- Place a Close VI from the CMC >> Instrument Control Palette onto the Block Diagram.
- Wire the **Instrument** tunnel of the Case Structure to the **Instrument** input terminal of the Instrument Control Close VI.
- Wire the **Error** tunnel of the Care Structure to the **error in (no error)** terminal of the Instrument Control Close VI.

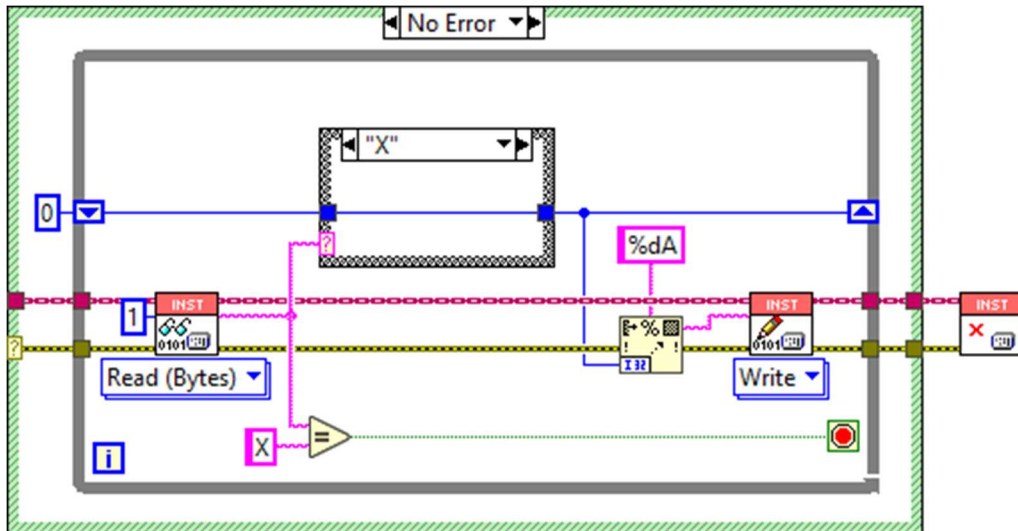


Figure 41: Processor Main Loop Close

5.3.3. Running UI and Processor

Now that both the UI and the Processor are complete, they can be run. Since the Processor is a listener, it will need to be started before the UI. This is to ensure that the UI has something to connect to.

6. Troubleshooting Recommendations

6.1. Console Refuses to Inject into a connection that has active communication.

The Console reports “Unable to Injection into a closed connection” when injection is attempted, but the Console also reports ongoing communication with the device.

This can happen if the user doesn't call Instrument Control.**Open**. This VI tells the Console that the connection is active. If the Console hasn't been told that the connection is active, it assumes that it is closed and will not attempt to inject.

6.2. Additional Resources.

See the NI forums user group for the Driver Framework. <https://forums.ni.com/t5/CMC-Driver-Framework/gp-p/5394>