UDP Data Link Custom Device

Customers need channel data in NI VeriStand to be reflected over the network via UDP data in a manner that can be received by a custom system. Reflective memory is not desirable due to cost, and its performance is unnecessary. NI VeriStand’s normal method for distributing data goes through a Gateway and is then available through a .NET API. This method is only available on Windows based operating systems. The general purpose nature of a data link via UDP will lead to a lot of re-use for National Instrument customers. This document covers the customer’s needs to provide a solution for their system and the needs of National Instruments to provide a general purpose custom device.

Contents

[Project Requirements 2](#_Toc295842224)

[Technical Requirements 2](#_Toc295842225)

[Application Requirements 2](#_Toc295842226)

[User interface requirements 3](#_Toc295842227)

[Functional requirements 8](#_Toc295842228)

[Data structure requirements 13](#_Toc295842229)

[Timing requirements 15](#_Toc295842230)

[Error handling requirements 16](#_Toc295842231)

[Initialization, shutdown requirements 17](#_Toc295842232)

[Project Organization and Hierarchy 19](#_Toc295842233)

[LabVIEW Project hierarchy 19](#_Toc295842234)

[Disk hierarchy 20](#_Toc295842242)

[Paths 20](#_Toc295842245)

# Project Requirements

Create a mechanism for communicating data in a high-speed, low-latency fashion from and to a system running NI VeriStand’s execution engine.

## Technical Requirements

Determine and list the following requirements from the project specification.

### Application Requirements

Goal and purpose of the application

APP1: Configuration

The custom device must have the ability for the user to configure it prior to execution.

APP1.1: UDP Configuration

The custom device must have the ability for the user to configure the UDP settings including: local IP address, remote IP address, and port number.

APP1.2: Tx Channels

The custom device must have the ability for the user to configure which channels in the system will be sent out to a remote system via UDP. These channels are referred to as Tx Channels.

APP1.3: Rx Channels

The custom device must have the ability for the user to configure which channels in the system will be received from a remote system via UDP. These channels are referred to as Rx Channels.

APP2: Packet Information

The custom device must have a mechanism for generating a text file that describes the packet information for both Tx and Rx packets. The custom device may generate the file on compile of the system definition. This file will be used by the external system to communicate with the NI VeriStand system.

APP3: Execution

This custom device should send a Tx packet (UDP) containing Tx channels at the configured rate of the custom device and NI VeriStand, and it should be capable of receiving an Rx packet (UDP) containing Rx channels at the same rate that it sends Tx packets.

App4: Performance

This custom device does not have to meet any specific performance criteria, but it must have as small a footprint as possible and not interfere with deterministic execution of the rest of the system.

App4.1: System Channel Duplication

This custom device should not duplicate any channels unnecessarily in the system in order to optimize performance. It may duplicate Rx channels if necessary to meet VeriStand’s architecture.

App4.2: Rx and Tx Channel Access

This custom device should access Rx and Tx channels in a deterministic low-latency fashion.

App4.3: UDP Communication

This custom device should perform all UDP communication in a parallel process that will not impact system jitter and determinism if errors or network slow-downs occur.

### User interface requirements

Presentation and behavior of controls that interact with users. This custom device will be used as a general purpose custom device so help, icons, right-click menus, and buttons should be used to aid usability and decrease long-term support of this custom device.

UI1: UDP Data Link Custom Device

The custom device must appear in System Explorer’s tree if added by a user.

UI1.1: UDP Data Link Main Page

The custom device must have a UI page that appears when the custom device is selected in the tree.

UI1.1.1: Documentation

The main page must have documentation that describes how to use and configure the custom device built into its main page.

UI1.1.2: Local IP address

The main page must have a control that allows the user to specify the IP address of the local system’s NIC to send and receive UDP packets. The control should be named “IP Address of Local NIC”.

UI1.1.3: Auto-Assign Local IP Address

The main page must have a checkbox control to auto-assign the local IP address control to the IP address configured to the target in System Explorer. When the checkbox is enabled, the Local IP Address control should disable and display the IP address configured for the target system.

UI1.1.4: Remote System IP Address

The main page must have a control that allows the user to specify the IP address of the remote system that will be communicated with. The control should be named “IP Address of Remote Machine”.

UI1.1.5: Port Number

The main page must have a control that allows the user to specify the port number that is communicated on for both sending and receiving UDP packets

UI1.1.6: Decimation

The main page must have a control that allows the user to specify the decimation of the custom device that dictates the rate at which the custom device executes.

UI1.1.7: Timestamp

The main page must have a control that allows the user to specify a channel used for timestamp information by the custom device. The control should be initialized to the system time channel.

UI1.1.8: Export to File Checkbox

The main page must have a checkbox control that allows the user to specify whether the channel list is exported to file on system definition compilation. If checked, the control should enable a path control so that the user can specify a file path to save the exported file.

UI1.2: Export Dynamic Button

The custom device may have an export dynamic button that is displayed when the custom device is selected and generates a channel list file if pressed.

UI1.3: Import Dynamic Button

The custom device may have an import dynamic button that is displayed when the custom device is selected and generates channel configuration for the custom device.

UI1.4: Export Run-Time Menu

The custom device may have an export run-time menu that is activated when right-clicking the custom device and generates a channel list file when selected.

UI1.5: Import Run-Time Menu

The custom device may have an import run-time menu that is activated when right-clicking the custom device and generates channel configuration for the custom device when selected.

UI2: Tx Channels

The custom device must have a section for Tx Channels.

UI2.1: Tx Channels UI Page

The Tx Channels section must have a UI page that appears when the section is selected in the tree.

UI2.1.1: Documentation

The Tx page should have documentation that describes how to configure the Tx channels.

UI2.1.2: Channel List

The Tx page should list all the Tx channels previously selected by the user in a list control.

UI2.1.2.1: Reorder Drag and Drop

The list control may support drag and drop functionality for adjustment of the transmit order of the list of Tx channels.

UI2.2: Add Channels Dynamic Button

The Tx Channels section should have an add channels dynamic button that is displayed when the Tx Channels section is selected. A dialog to choose Tx Channels to add should appear if the button is pressed.

UI2.3: Remove Channels Dynamic Button

The Tx Channels section should have a remove channels dynamic button that is displayed when the Tx Channels section is selected. A dialog to choose Tx Channels to remove should appear if the button is pressed.

UI2.4: Add Channels Run-Time Menu

The Tx Channels section should have an add channels run-time menu that is activated when right-clicking the Tx Channels section. A dialog to choose Tx Channels to add should appear if the menu item is selected.

UI2.5: Remove Channels Run-Time Menu

The Tx Channels section should have a remove channels run-time menu that is activated when right-clicking the Tx Channels section. A dialog to choose Tx Channels to remove should appear if the menu item is selected.

UI3: Rx Channels

The custom device must have a section for Rx Channels.

UI3.1: Rx Channels UI Page

The Rx Channels section must have a UI page that appears when the section is selected in the tree.

UI3.1.1: Documentation

The Rx page should have documentation that describes how to configure the Rx channels.

UI3.1.2: Channel List

The Rx page should list all the Tx channels previously selected by the user in a list control.

UI2.1.2.1: Reorder Drag and Drop

The list control may support drag and drop functionality for adjustment of the transmit order of the list of Rx channels.

UI3.2: Add Channels Dynamic Button

The Rx Channels section should have an add channels dynamic button that is displayed when the Rx Channels section is selected. A dialog to choose Rx Channels should appear if the button is pressed.

UI3.3: Remove Channels Dynamic Button

The Rx Channels section should have a remove channels dynamic button that is displayed when the Rx Channels section is selected. A dialog to choose Rx Channels to remove should appear if the button is pressed.

UI3.4: Add Channels Run-Time Menu

The Rx Channels section should have an add channels run-time menu that is activated when right-clicking the Rx Channels section. A dialog to choose Rx Channels should appear if the menu item is selected.

UI3.5: Remove Channels Run-Time Menu

The Rx Channels section should have a remove channels run-time menu that is activated when right-clicking the Rx Channels section. A dialog to choose Rx Channels to remove should appear if the menu item is selected.

UI3.6: Rx Channel

The Rx Channels section must have a channel in the tree for each user selected Rx Channel.

UI3.6.1: Rx Channel UI Page

The Rx Channel must have a UI page that appears when the channel is selected in the tree.

UI3.6.1.1: Name

The Rx Channel name must be shown on the UI page. The Rx Channel must have the same name as the channel selected.

UI3.6.1.2: Description

The Rx Channel description must be shown on the UI page. The Rx Channel should have the same description as the channel selected.

UI3.6.1.3: Channel Destination

The Rx Channel UI page should display the path to the channel that was selected by the user.

UI4: Add Channels Dialog

This is the dialog that is activated when adding either Tx or Rx channels. This dialog must allow a user to select channels to include as Tx Channels. An example of what the dialog may look like:

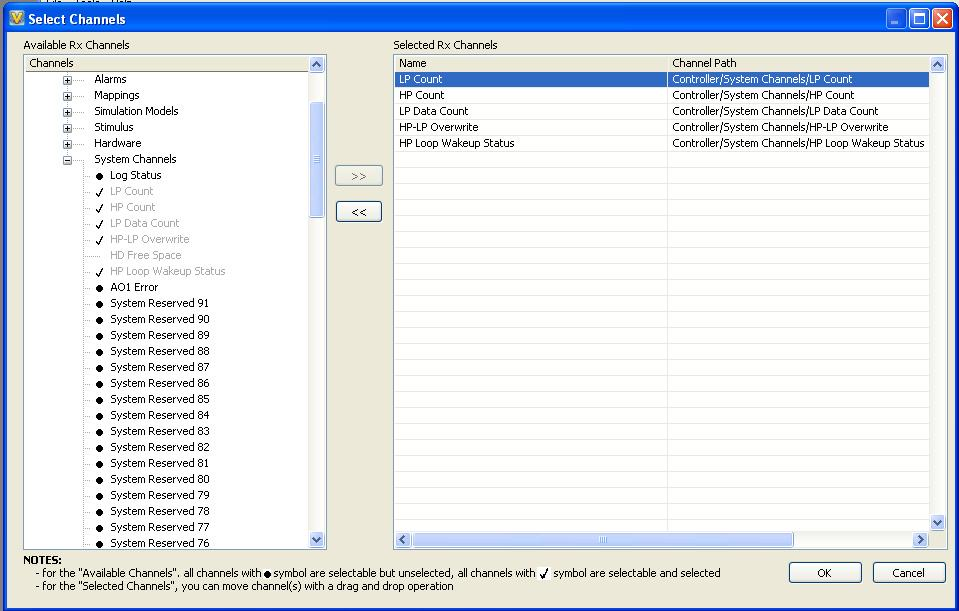


Figure 1: Add Channels Dialog for Tx and Rx channels

UI4.1: Channel List

The dialog should contain a list of the channels already added by the user.

UI4.2: Re-order Items

The list control should support drag and drop functionality for adjustment of the transmit or receive order of the list of channels.

UI4.3: Remove Channels

The dialog may support the ability for the user to directly remove channels from the dialog.

UI4.4: Available Channels Tree

The tree dialog that displays the list of channels in the system should closely resemble the standard .NET tree control that NI VeriStand provides in order to maintain consistency.

UI5: Remove Channels Dialog

This is the dialog that is activated when removing Tx or Rx channels. This dialog must allow a user to select channels to remove. The dialog should match the look and feel of the other remove multiple item dialog built into NI VeriStand.

UI6: Status Channels

The custom device must have a section for Status Channels. This section may include: Last Error, Last Error Timestamp, Tx Count, and Rx Count.

UI6.1: Documentation

The Status Channels section must have documentation built into it which describes each of the status channels.

### Functional requirements

The functionality of the components and their interaction within the system

FR1: System Explorer Configuration

Inputs: User Input

Behavior: Sets the configuration settings for the UDP communication, Rx packet, and Tx packet for the system definition and generates an XML definition of the UDP Data Link.

Output: System Definition with configured UDP Data Link

FR1.1: UDP Data Link Custom Device Config

Inputs: User input for Local IP Address, Remote IP Address, Port Number, Decimation, Timestamp channel

Behavior: Sets the UDP and execution properties for execution of the custom device.

Outputs: Modified System Definition containing new custom device settings

FR1.2: Tx Channel Config

Inputs: User Input

Behavior: Sets the Tx channel dependencies for execution from System Explorer.

Outputs: Modified System Definition containing configuration for the Tx channels.

FR1.2.1: Add Tx Channel(s)

Inputs: User selected Add Tx Channel(s) and selected Channel Paths to add

Behavior: Adds dependencies to channels in the system whose data will be sent in the Tx Packet. This adds channels to the Tx channel list.

Outputs: Modified System Definition containing additional Tx channels.

FR1.2.2: Remove Tx Channel

Inputs: User selected Remove Tx Channel and selected Tx Channel to remove

Behavior: Removes dependency to the Tx channel in the system that was going to be sent in the Tx Packet. This removes the channel from the Tx channel list.

Outputs: Modified System Definition with a Tx channel removed.

FR1.2.3: Update Tx Channel List on Save

Inputs: User selected save

Behavior: Checks all Tx channel dependencies on save to make sure the channels were not removed from the system. If channels were removed from the system, the dependencies of those channels are removed from the Tx channel list.

Outputs: System Definition without any missing Tx channel dependencies from the UDP Data Link custom device

FR1.3: Rx Channel Config

Inputs: User Input

Behavior: Sets the Rx channel dependencies for execution from System Explorer.

Outputs: Modified System Definition containing configuration for the Rx channels

FR1.3.1: Add Rx Channel(s)

Inputs: User selected Add Rx Channel(s) and selected Channel Paths to add

Behavior: Adds local Rx channels to the Rx Channels section and creates a mapping from the local channel to the channel from the system that was input. This adds channels to the Rx channel list. The local channels will be written to from the Rx packet.

Outputs: Modified System Definition containing additional Rx channels.

FR1.3.2: Remove Rx Channel

Inputs: User selected Remove Rx Channel and selected Rx Channel to remove

Behavior: Removes local Rx channel from the Rx Channel section and removes the mapping from the local channel to the corresponding channel in the system. This removes a channel from the Rx channel list.

Outputs: Modified System Definition with an Rx channel removed.

FR1.3.2.1: Remove Rx Channel Mapping

Inputs: Rx Channels

Behavior: Removes the mapping from the local Rx channels to the corresponding channels in the system.

Outputs: Modified System Definition with an Rx mapping removed

FR1.3.3: Update Rx Channel List on Save

Inputs: User selected save

Behavior: Checks all Rx channel dependencies on save to make sure the mapped channels were not removed from the system. If any mapped channels were removed from the system, the mapping is removed and the local Rx channel is deleted from the Rx channel list.

Outputs: System Definition without any missing Rx channel dependencies or mappings from the UDP Data Link custom device

FR1.3.3.1: Update Rx Channel Mapping

Inputs: User Input: Update Mapping?

Behavior: If a mapping from a local Rx channel to the channel in the system is removed, on save of the system definition the user will be prompted for how they would like to handle the situation. If the user would like to remap, the mapping is recreated. If the user does not want to remap, the local Rx channel is deleted.

Outputs: System Definition without any missing Rx channel mappings from the UDP Data Link custom device

FR1.4: Delete Custom Device

Inputs: User selected delete custom device

Behavior: Calls Remove Rx Channel Mapping for all Rx channels in the custom device so that no hanging mappings are left over when the custom device is gone.

Outputs: System Definition with no trace that the UDP Data Link custom device was ever added.

FR1.5: Export Packet Definition File

Inputs: User selected Export and entered file path

Behavior: Generates a Packet Definition File according to the XML Data Structure which contains the system definition’s current Rx and Tx channels, controller name, version number, and any other data required for the file.

Outputs: XML File

FR1.6: Export on Compile

Inputs: User compiles System Definition, Export to File

Behavior: If Export to File is true, calls Export Packet Definition File

Outputs: XML File

FR1.7: Import Packet Definition File

Inputs: User selected Import and entered path to XML File

Behavior: When a user initiates an import, the custom device should read the Rx and Tx channels from a Packet Definition File and add the channels to the Tx and Rx channel list for all the channel paths that are valid for the current system definition file. Any conflicts should be reported to the user and handled according to user input.

Outputs: Modified System Definition with Tx and Rx Channels added to system

FR1.7. 1: Import Channels Already in Channel List

Inputs: Channels that are already added to the channel lists

Behavior: If some channel paths are already included as Rx and Tx channels, the user should be notified that some channels were already added.

Outputs: User Dialog

FR1.7.2: Import Channel Paths Invalid

Inputs: Channels that do not exist in the system

Behavior: If some channel paths are not valid, the user should be notified that some channel were unable to be added because they do not exist.

Outputs: User Dialog

FR1.7.3: Import Rx Channels with Mappings

Inputs: Channels that already have mappings, User Input

Behavior: If any Rx channels are specified that already have mappings applied, the user should be prompted on how to handle.

Outputs: System Definition that may or may not have updated channel mappings

FR2: Real-Time Execution

Inputs: System Definition Configuration

Behavior: The Real-Time execution reads/writes channel data in an Inline Hardware Custom Device VI and sends Tx packets and receives Rx packets in a Asynchronous UDP loops. The two VIs run with the settings configured in the System Definition configuration.

Outputs: Periodic Tx Packets to remote machine, Rx channel data to NI VeriStand system

FR2.1: Inline Hardware Custom Device

Inputs: Rx Channel list, Tx Channel list, and Status Channel list, Rx UDP data, Status UDP data, Tx Channel data

Behavior: At the rate configured in the system definition, the Inline Hardware Read case is executed at the beginning of NI VeriStand’s primary control loop to apply UDP Rx data to Rx Channels, and the Inline Hardware Write case is executed at the end of NI VeriStand’s primary control loop to apply Tx Channels to UDP Tx data.

Outputs: Tx Channel data to UDP RT FIFO, Rx UDP data to NI VeriStand’s Rx channels

FR2.1.1: Inline Hardware Read

Inputs: Rx Channel list, Status Channel list, Rx UDP data, Status UDP data

Behavior: At the rate configured in system definition and as defined in the timing requirements, this case executes, reads Rx and Status data from the Rx UDP loop via RT FIFO, and writes it to NI VeriStand’s Rx channels and Status channels. Timing reporting is also performed as defined in the timing requirements.

Outputs: Rx UDP data to NI VeriStand’s Rx channels

FR2.1.2: Inline Hardware Write

Inputs: Tx Channel list, Tx Channel data

Behavior: At the rate configured in system definition and as defined in the timing requirements, this case executes, reads Tx data from NI VeriStand’s Tx channels, and writes them to the Tx UDP loop via RT FIFO. Timing reporting is also tracked as defined in the timing requirements so that the Inline Hardware Read may update.

Outputs: Tx Channel data to UDP RT FIFO

FR2.2: Parallel UDP Loops

Inputs: UDP Settings, Packet Header Info, Tx Channel data, Rx Packet from network

Behavior: The parallel UDP loops set up the UDP connection as specified by the UDP settings from the System definition. Tx data is sent through the UDP Tx Loop, and Rx data is received through the UDP Rx Loop.

Outputs: Rx UDP data to Inline VI, Tx Packet to network

FR2.2.1: UDP Tx Loop

Inputs: UDP connection, Packet Header Info, Tx Channel data

Behavior: This loop waits on Tx Channel data from the inline VI that is received via RT FIFO. It then packages the data and the packet header info into a Tx packet and sends it out on the open UDP session.

Outputs: Tx packet to network

FR2.2.2: UDP Rx Loop

Inputs: UDP connection, Packet Header Info, Rx Packet from network

Behavior: This loop waits on a UDP packet on the UDP connection. When it receives the packet, it extracts the Rx channel data out of the packet, determines any errors as specified in the error requirements, packages all the data and sends the data to the inline VI via RT FIFO.

Outputs: Rx UDP data to Inline VI

### Data structure requirements

Definition of any data structure requirements for the system

DSR1: Packet Data Structure

The Tx and Rx packets must have a defined packet header followed by an array of defined channel data types with a defined endian scheme. Each Tx or Rx packet must begin with the packet header and must transmit all Tx or Rx channels in the corresponding packet. Packets may not have partial channel data.

DSR1.1: Endian

Individual packet and channel data elements will be sent with a little-endian scheme.

DSR1.2: Packet Header

The packet header must be defined by the following pieces of information and their data type.

* Packet Size U16
  + This is the number of bytes in the corresponding packet. It includes both packet header and channel data when calculating number of bytes.
* Protocol ID U8
  + Used to track the revision of the Packet Data Structure’s definition. Should statically be set to a value of 1.
* Protocol Revision U8 (4 MSB for major revision, 4 LSB for minor revision)
  + Used to track the revision of the Packet Data Structure’s definition. Should statically be set to a value of binary: 00010000.
* Sequence Number U32
  + For Tx packets, this is the packet number sent. It is 0 indexed.
  + For Rx packets, this is the Tx packet number that the Rx packet is responding to.
* System Definition Version U16 [Major: (12:15), Minor: (8:11), Fix: (4:7), Build (0:3)]
  + This is the version of the system definition file as found in System Explorer
* Source Node Name Char[32]
  + For Tx packets, this is the name of the NI VeriStand controller defined in the system definition.
  + For Rx packets, the source node name is in the packet header but not used by the custom device.
* Timestamp DBL (64-bit floating point number)
  + For Tx packets, this is the value of the timestamp channel configured in the custom device.
  + For Rx packets, the data Timestamp value is in the packet header but not used by the custom device.

DSR1.3: Channel Data Type

Each Tx or Rx channel’s data will be sent as a 32-bit floating point numeric.

DSR2: XML Data Structure

The data structure for XML used in export and import must have the following definition. There may be a schema definition developed.



Figure 2: XML template for exported UDP Data Link information

Where identification dataRevision, identification creationDate, controller name, group freq, channel name, channel description, and channel unit are values that are determined by the system definition. A channel tag will exist for each corresponding Tx or Rx channel.

* identification dataRevision
  + Major, Minor, Fix, and Build are all integer values that will correspond to the system definition’s version. The dataRevision corresponds directly to the “System Definition Version” item in the Rx and Tx packet headers.
* identification creationDate
  + YYYY, MM, and DD are all integer values and they correspond to the date in which the file was created.
* controller name
  + This is the name of the controller in the system definition that the custom device resides under. The controller name corresponds directly to the “Source Node Name” in the Rx packet header.
* group freq
  + Hz is an integer value that corresponds to the execution rate of the custom device as defined by the controller rate and the custom device’s decimation. It is the data transmission frequency of the custom device.
* channel name
  + This is the individual channel path of a Tx or Rx channel. For Rx channels, the path is specified for the item that the local channel is mapped to.
* channel description
  + This is the individual channel description for the same channel as described in channel name.
* channel unit
  + This is the individual channel unit for the same channel as described in channel name.
* channel dataType
  + The dataType should always be set to SGL as described in the packet definition.

### Timing requirements

Hardware / software, event-based, resolution, jitter, data overflow, daylight savings

TR1: Timing Rate

The custom device must be timed by a decimated rate of the Primary Control Loop. The decimation factor is a positive integer {1, 2, 3, …} specified by the user interface.

TR2: Packet Scheduling

A Rx packet corresponding to the response of Tx (n-1) packet should be applied to channels one Primary Control Loop iteration after Tx (n) packet was scheduled to transmit. Tx packets must not be scheduled to transmit at any time other than the end of a Primary Control Loop iteration. Rx packets must not be applied to channel data at any time other than the beginning of a Primary Control Loop iteration. See figures below.

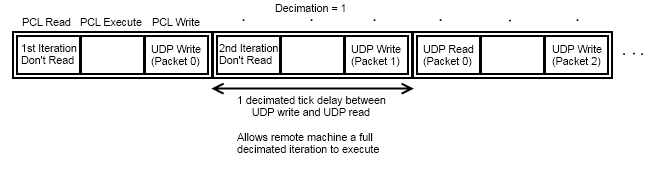


Figure 3: Packet Scheduling and Latency with Decimation = 1

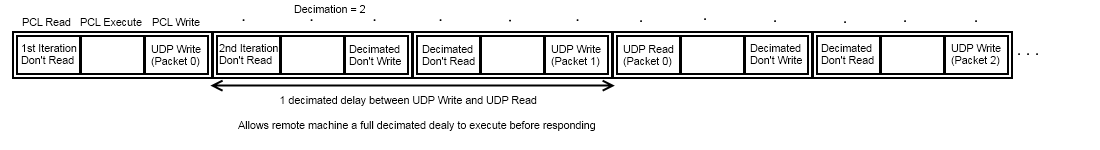


Figure 4: Packet Scheduling and Latency with Decimation = 2

TR3: Latency

The custom device must have a 1 cycle latency between Tx transmission and when the corresponding Rx received packet will be applied to the system’s channels. The 1 cycle latency is in terms of custom device execution rate and not in terms of NI VeriStand execution rate. See figures for packet scheduling.

TR4: Timing Reporting

The custom device must report all underflows of the Rx packets and overflows of Tx packets being sent.

### Error handling requirements

Warning, errors, critical errors, shutdown sequence

ERR1: Execution Critical Errors

Any error on initialization, start, reading channel data from NI VeriStand, writing data to NI VeriStand, or errors that indicate that the RT FIFOs to the UDP loops have been destroyed shall be considered critical errors. All critical errors must halt NI VeriStand and report the error to the console.

ERR1.1: UDP Loops Critical Errors

If a critical error is encountered in one UDP loop, it must be able to stop both UDP loops and signal the critical error to the inline VI.

ERR2: Execution Critical Error Cleanup

On a critical error, the custom device must still try to destroy any references and free any memory it may have consumed. NI VeriStand should be in a state able to accept a re-deployment of a system definition after a critical error occurs.

ERR3: Execution UDP Error Reporting

All UDP errors and warnings should be captured and reported to NI VeriStand via Status Channel. A timestamp of when the error occurred should also be captured and report to NI VeriStand via Status Channel.

ERR3.1: UDP Error Types

The Rx UDP loop must define the following non-critical error conditions in order of importance. If multiple errors occur, the highest priority error must be the one reported.

ERR3.1.1: UDP Timeout Error

This is the error 56 error code that can be returned if the UDP Read times out before receiving data. If error 56 occurs, the custom device should not report that a new packet was received.

ERR3.1.2: Protocol Error

Error -5000 should be reported if the protocol id or revision received in the Rx packet does not match what the custom device expects. If a protocol error occurs, the custom device should report that a packet was received, but the Rx channel data should not be updated.

ERR3.1.3: Packet Size Error

Error -5001 should be reported if the packet size does not match value of the packet size in the packet header or does not match the expected packet size based on expected packet header size and expected Rx channel data size. If a packet size error occurs, the custom device should report that a packet was received, but the Rx channel data should not be updated.

ERR3.1.4: System Definition Version Error

Error -5002 should be reported if the system definition version does not match the value for the system definition version in the packet header. If a system definition version error occurs, the custom device should report that a packet was received, but the Rx channel data should not be updated.

ERR3.1.5: Sequence Number Error

Error -5003 should be reported if the sequence number in the packet header of the Rx packet does not match the sequence number in the most recently sent Tx packet. If a sequence number error occurs, the custom device should report that a packet was received, and the channel data should also be updated.

ERR4: Execution Packet Error Reporting

All Packet errors should be captured and reported to NI VeriStand via Status Channel. A timestamp of when the error occurred should also be captured and report to NI VeriStand via Status Channel.

ERR5: Execution Warnings

Any warnings that are not associated with UDP connections are lost and not reported to the system.

ERR6: Configuration Errors

All errors encountered in System Explorer associated with the custom device will be displayed to the user via an error dialog.

### Initialization, shutdown requirements

User interface and program behavior during startup, error conditions, and shutdown

ISR1: Configuration Initialization

When added to the system definition, the custom device must initialize.

ISR1.1: Initialize UDP

The custom device must initialize the UDP settings with: auto-assign local IP: true, local IP: same IP as target’s IP, remote IP: 192.168.0.1, port: 51936, decimation: 1, timestamp: System Time, and export file checkbox: true.

ISR1.2: Initialize Rx Channels

The custom device must create a section for Rx Channels on initialization.

ISR1.3: Initialize Tx Channels

The custom device must create a section for Tx Channels on initialization.

ISR1.4: Initialize Status Channels Section

The custom device must create a section for Tx Channels on initialization.

ISR1.4.1: Initialize Status Channels channels

The custom device must create channels for each status channel under the Status Channels section on initialization.

ISR2: Real-Time Execution Initialization

When deployed, the custom device must initialize.

ISR2.1: Channel Reference Initialization

Initialization must get Tx, Rx, and Status channel references and data buffers for data access.

ISR2.2: Rx and Tx RT FIFOs

Initialization must create RT FIFOs for transferring data from inline VI to the parallel UDP process. The RT FIFOs should have a depth of 1 element. The data type of the RT FIFO should be an array of the data type of channels (DBL) and should specify the number of elements in each RT FIFO according to the number of channel references initialized.

ISR2.3: Initialize Packet Header Information

Initialization must get packet header information from system definition for use in UDP loops.

ISR2.4: Launch UDP Loops in parallel process

Initialization must launch UDP loops for execution in a parallel process. UDP loops should accept Rx Channel initial values and references to the Tx and Rx RT FIFOs.

ISR2.4.1: Signal UDP Loops Started

The UDP loops should signal to the inline VI that they have started by writing an empty array to the Rx RT FIFO. The inline VI should wait to start until it receives the signal from the UDP loops. If it does not receive the signal within 5 seconds, a critical error should be set.

ISR3: Real-Time Execution Shutdown

Before execution stops, the custom device must perform a shutdown sequence. The shutdown sequence must close references to all data transfer mechanisms (i.e. RT FIFOs, Notifiers, etc.).

ISR3.1: UDP Shutdown

If a critical error did not shutdown the UDP loops, they should be shut down by destroying the communication mechanism (i.e. RT FIFO) to those loops. The UDP loops should stop on error received after the communication mechanism is destroyed.

ISR3.2: Normal Shutdown

On a normal shutdown sequence not due to a critical error, the errors should be read from the UDP loop, and then the communication mechanism should be destroyed. After destroying the communication mechanism, the custom device should wait on synchronization from the UDP loops signaling that they have shutdown and then destroy the synchronization and error reporting mechanisms.

# Project Organization and Hierarchy

## LabVIEW Project hierarchy

Develop a LabVIEW Project hierarchy for development

### **Custom Device Library**

The custom device must be contained within a LabVIEW Project Library.

#### Hierarchy

The project library must contain System Explorer and Real-Time virtual folders. It may also contain Debug Utilities. Any TypeDefs shared between System Explorer and Real-Time must be placed in a virtual folder named TypeDefs at the top-level of the project library’s hierarchy.

#### System Explorer Hierarchy

System Explorer virtual folder may contain Actions, Glyphs, Pages, Run-Time Menus, TypeDefs, and Utilities virtual folders. The corresponding VIs must be placed within these virtual folders. Any modules developed for use in the system explorer must be placed within the Utilities virtual folder. Initialization VI must be placed at the top-level of the System Explorer hierarchy.

#### Real-Time Hierarchy

Real-Time virtual folder may contain Utilities and TypeDefs virtual folders. Any modules developed for use in the real-time execution must be placed within the Utilities virtual folder. RT Driver VI must be placed at the top-level of the Real-Time hierarchy.

### Custom Device API Libraries

The custom device API, offline API, and utility libraries must be added to the project.

### Shared subVI / custom controls

Any shard libraries, subVIs, or custom controls used by the custom device library must be added to the project in a virtual folder labeled “shared”. If a shared item is not installed with LabVIEW or NI VeriStand and not available via download, the shared item should be copied to the local project directory.

### Source Only VIs

If possible, all VIs of the custom device library should be set to source only from the project properties to promote source code control and reduce file size on disk.

### Documentation Files

Any built documentation files should be contained within the custom device library under the System Explorer.

### Specify a naming convention

All pages must end in “Page.vi”. All actions must end in the name of action (i.e. On Compile Action.vi). All run-time menus must end in “RTM.vi”. All run-time menu dependencies must end in “RTM Dependency.vi”. The project, custom device library, RT driver VI, and initialization VI must all begin with the name of the custom device (i.e. “UDP Data Link RT Driver VI.vi”).

### Source Distribution Builds

Source distribution builds may be built from the My Computer target. In some cases it may be necessary to build the engine LLB from an RT target. If possible there should be an engine LLB build under a VxWorks target in order to provide VxWorks support in addition to PharLap. All source distributions should attempt to optimize settings such as disable debugging, remove block diagram, and remove front panel if possible.

## Disk hierarchy

Develop a disk hierarchy mechanism for development

### Mimic project hierarchy on disk

The custom device’s library hierarchy shall be mimicked on disk. The project file, library file, and all direct children shall be located at the top-level source-code directory. All other files will mimic the child directory structures.

### Documentation Files

The individual files to create any built documentation should be contained within a directory named “Help Project” at the top-level source code directory.

## Paths

Develop a path convention for maintainability in support and development

### Use (relative) symbolic paths

All VIs of the custom device will be built into an LLB for relative pathing.

### Utilization of relative and absolute path in code

All VIs called dynamically should be referenced relative to the LLB path. Any dependent file that was copied through NI VeriStand’s file dependency should use NI VeriStand’s API call to determine the path of the file.