# Customizing Protocol Descriptions for Packet Viewer

Online Help

KEYSIGHT
TECHNOLOGIES

# Notices

## Trademarks

Microsoft®, MS-DOS®, Windows®, Windows 2000®, and Windows XP® are U.S. registered trademarks of Microsoft Corporation.

## Revision History

August 2014

Available in electronic format only

Keysight Technologies
1900 Gardens of the Gods Road
Colorado Springs, CO 80907 USA

## Warranty

THE MATERIAL CONTAINED IN THIS DOCUMENT IS PROVIDED "AS IS," AND IS SUBJECT TO BEING CHANGED, WITHOUT NOTICE, IN FUTURE EDITIONS. FURTHER, TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, KEYSIGHT DISCLAIMS ALL WARRANTIES, EITHER EXPRESS OR IMPLIED WITH REGARD TO THIS MANUAL AND ANY INFORMATION CONTAINED HEREIN, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. KEYSIGHT SHALL NOT BE LIABLE FOR ERRORS OR FOR INCIDENTAL OR CONSEQUENTIAL DAMAGES IN CONNECTION WITH THE FURNISHING, USE, OR PERFORMANCE OF THIS DOCUMENT OR ANY INFORMATION CONTAINED HEREIN. SHOULD KEYSIGHT AND THE USER HAVE A SEPARATE WRITTEN AGREEMENT WITH WARRANTY TERMS COVERING THE MATERIAL IN THIS DOCUMENT THAT CONFLICT WITH THESE TERMS, THE WARRANTY TERMS IN THE SEPARATE AGREEMENT WILL CONTROL.

## Technology Licenses

The hardware and/or software described in this document are furnished under a license and may be used or copied only in accordance with the terms of such license.

## Restricted Rights Legend

If software is for use in the performance of a U.S. Government prime contract or subcontract, Software is delivered and licensed as "Commercial computer software" as defined in DFAR 252.227-7014 (June 1995), or as a "commercial item" as defined in FAR 2.101(a) or as "Restricted computer software" as defined in FAR 52.227-19 (June 1987) or any equivalent agency regulation or contract clause. Use, duplication or disclosure of Software is subject to Keysight Technologies' standard commercial license terms, and non-DOD Departments and Agencies of the U.S. Government will receive no greater than Restricted Rights as defined in FAR 52.227-19(c)(1-2) (June 1987). U.S. Government users will receive no greater than Limited Rights as defined in FAR 52.227-14 (June 1987) or DFAR 252.227-7015 (b)(2) (November 1995), as applicable in any technical data.

## Safety Notices

**CAUTION**

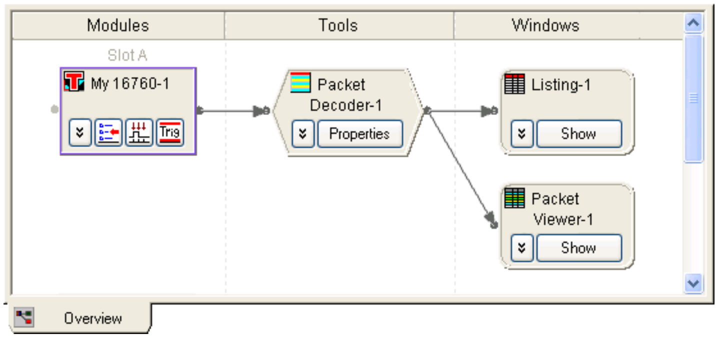
A CAUTION notice denotes a hazard. It calls attention to an operating procedure, practice, or the like that, if not correctly performed or adhered to, could result in damage to the product or loss of important data. Do not proceed beyond a CAUTION notice until the indicated conditions are fully understood and met.

**WARNING**

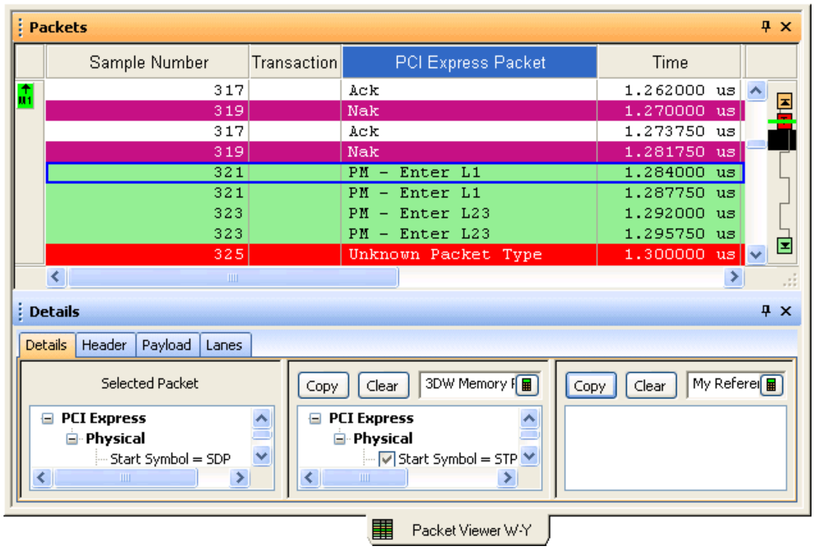
A WARNING notice denotes a hazard. It calls attention to an operating procedure, practice, or the like that, if not correctly performed or adhered to, could result in personal injury or death. Do not proceed beyond a WARNING notice until the indicated conditions are fully understood and met.

# Protocol Development Kit (PDK)—At a Glance

The *Keysight Logic Analyzer* application has a Packet Decoder tool that decodes captured logic analyzer data into packet information.



There is also a Packet Viewer window that displays decoded packet information.



| Sample Number | Transaction | PCI Express Packet | Time |
|---|---|---|---|
| 317 | | Ack | 1.262000 us |
| 319 | | Nak | 1.270000 us |
| 317 | | Ack | 1.273750 us |
| 319 | | Nak | 1.281750 us |
| 321 | | PM - Enter L1 | 1.284000 us |
| 321 | | PM - Enter L1 | 1.287750 us |
| 323 | | PM - Enter L23 | 1.292000 us |
| 323 | | PM - Enter L23 | 1.295750 us |
| 325 | | Unknown Packet Type | 1.300000 us |

Both the Packet Decoder tool and the Packet Viewer window work with multiple protocols, specified by XML-format protocol description files.

Protocol description files describe the way a protocol is framed, decoded, encoded (prepared for the "Find a packet" trigger function), and displayed. You can edit existing protocol descriptions and add new ones. This document describes how.

A single protocol description file is created for each protocol family.

Protocol description files are loaded when the *Keysight Logic Analyzer* application starts or when "refreshed" in the Packet Decoder tool.

In the Packet Decoder tool's Properties dialog, you can select from the protocols that have been loaded.

- Chapter 1, “Editing an Existing Protocol Description,” starting on page 9
- Chapter 2, “Creating a New Protocol Description,” starting on page 17
- Chapter 3, “Using Formulas,” starting on page 55

- Chapter 4, “Solving Problems,” starting on page 61
- Chapter 5, “Multi-Lane Serial Link Concepts,” starting on page 67
- Chapter 6, “XML Element Reference,” starting on page 71
- Chapter 7, “Formula Reference,” starting on page 109

For more information on using the Packet Decoder tool, the Packet Viewer window, and the Event Editor, see:

- "Using the Packet Decoder Tool" (in the online help)
- "Analyzing Packet Data" (in the online help)
- "Using the Packet Event Editor" (in the online help) and "Event Editor Dialog" (in the online help)

For a printable version of this help file, see: "*Customizing Protocol Descriptions for Packet Viewer*".

# Contents

## 3 Using Formulas

## 4 Solving Problems

## 5 Multi-Lane Serial Link Concepts

## 6 XML Element Reference

7 Formula Reference

# 1 Editing an Existing Protocol Description

## Starting the Protocol Description File Editor

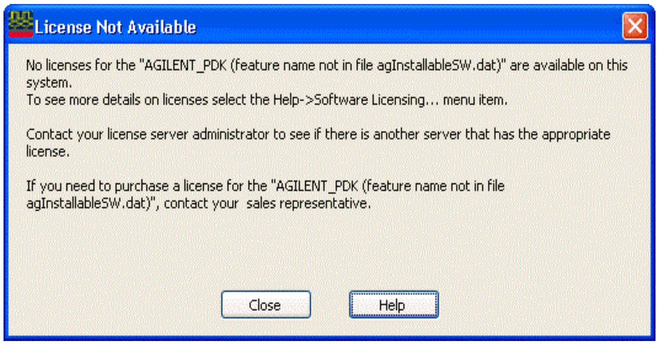
1 From the Windows Start menu, choose **Start>All Programs>Keysight Logic Analyzer>Keysight Protocol Development Kit**.

Or:

Click the **Keysight Protocol Development Kit** icon on the Windows Desktop.

**If No Licenses are Available**

If no licenses are available when starting the Protocol Development Kit (PDK), you get the following dialog:



No licenses for the "AGILENT_PDK (feature name not in file agInstallableSW.dat)" are available on this system.
To see more details on licenses select the Help->Software Licensing... menu item.

Contact your license server administrator to see if there is another server that has the appropriate license.

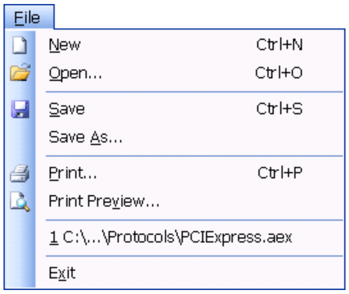
If you need to purchase a license for the "AGILENT_PDK (feature name not in file agInstallableSW.dat)", contact your sales representative.

You can get more information on licensing by choosing **Help>Software Licenses...** in the main *Keysight Logic Analyzer* application and by clicking the **Help** button in the resulting Software Licensing dialog.

(The **Help>Software Licenses...** menu item is also available in the protocol description file editor once it is licensed and you are able to start it.)

## Opening Protocol Description Files

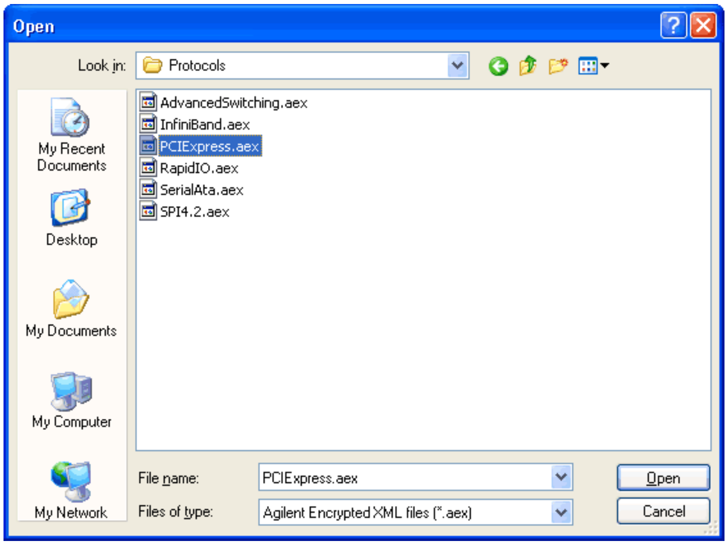
Protocol description files are opened (and saved) from the Keysight PDK (Protocol Development Kit) editor's **File** menu.



Protocol description files are located in the Protocols folder in the Keysight Logic Analyzer application's install directory. For example, the default location is: C:\Program Files\Keysight Technologies\Logic Analyzer\Protocols.

1 From the Protocol Description File Editor's main menu, choose **File>Open...**.

2 In the Open dialog, select the protocol description file you wish to open; then, click **Open**.



Look in: Protocols

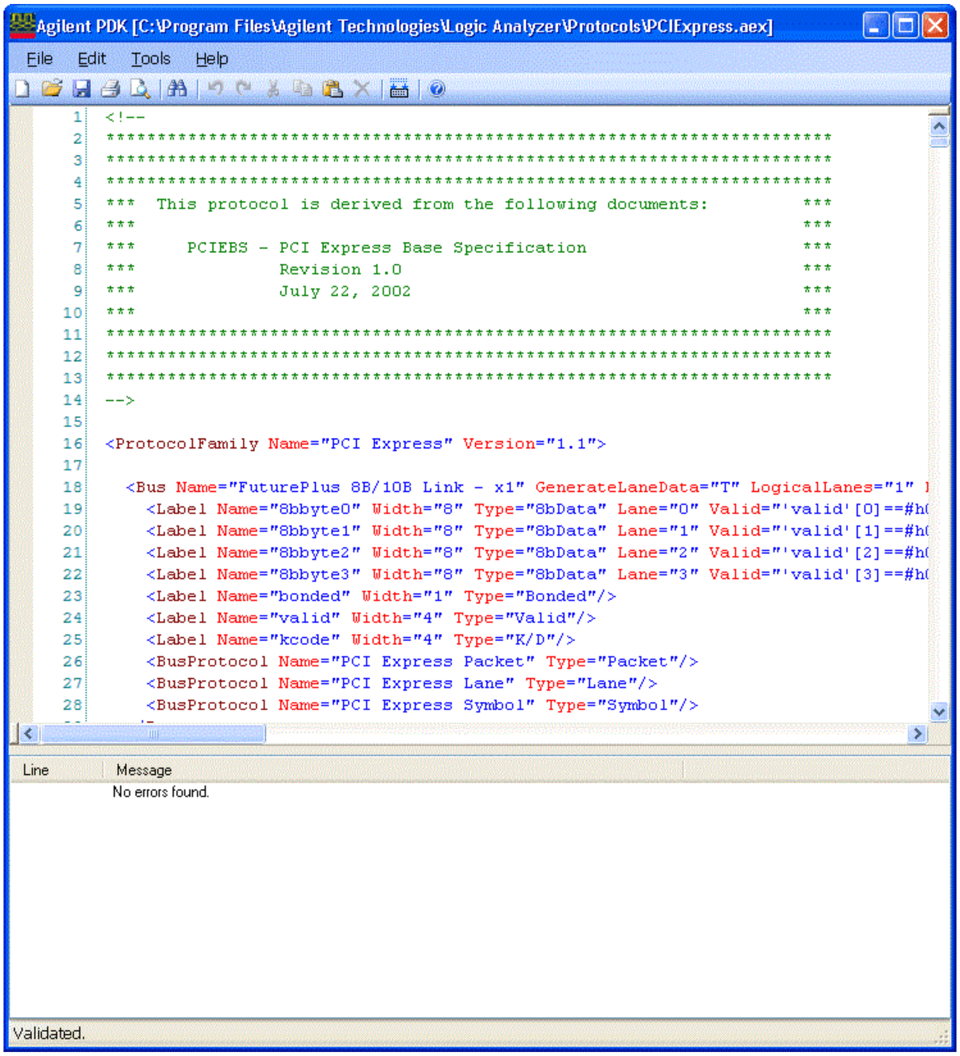
AdvancedSwitching.aex
InfiniBand.aex
PCIExpress.aex
RapidIO.aex
SerialAta.aex
SPI4.2.aex

File name: PCIExpress.aex

Files of type: Agilent Encrypted XML files (*.aex)

Protocol description files have the .aex (Keysight Encrypted XML) file extension.

The XML-format protocol description file appears in the PDK (Protocol Development Kit) editor window.


Agilent PDK [C:\Program Files\Agilent Technologies\Logic Analyzer\Protocols\PCIExpress.aex]
File
Edit
Tools
Help

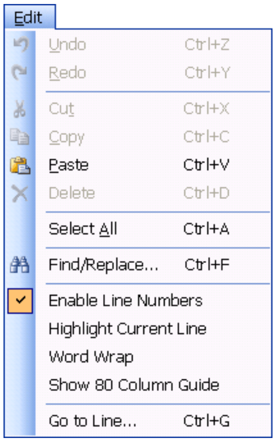
```
<!--
***********************************************************************
***********************************************************************
***********************************************************************
***  This protocol is derived from the following documents:         ***
***                                                                 ***
***     PCIEBS - PCI Express Base Specification                     ***
***              Revision 1.0                                       ***
***              July 22, 2002                                      ***
***                                                                 ***
***********************************************************************
***********************************************************************
***********************************************************************
-->

<ProtocolFamily Name="PCI Express" Version="1.1">

  <Bus Name="FuturePlus 8B/10B Link - x1" GenerateLaneData="T" LogicalLanes="1"
    <Label Name="8bbyte0" Width="8" Type="8bData" Lane="0" Valid="'valid'[0]==#h
    <Label Name="8bbyte1" Width="8" Type="8bData" Lane="1" Valid="'valid'[1]==#h
    <Label Name="8bbyte2" Width="8" Type="8bData" Lane="2" Valid="'valid'[2]==#h
    <Label Name="8bbyte3" Width="8" Type="8bData" Lane="3" Valid="'valid'[3]==#h
    <Label Name="bonded" Width="1" Type="Bonded"/>
    <Label Name="valid" Width="4" Type="Valid"/>
    <Label Name="kcode" Width="4" Type="K/D"/>
    <BusProtocol Name="PCI Express Packet" Type="Packet"/>
    <BusProtocol Name="PCI Express Lane" Type="Lane"/>
    <BusProtocol Name="PCI Express Symbol" Type="Symbol"/>
```

| Line | Message |
|---|---|
| | No errors found. |

Validated.

## Editing Protocol Description Files

Protocol description files are edited using the Keysight PDK (Protocol Development Kit) editor's **Edit** menu.



The PDK editor provides the standard text editing features:

- Undo and Redo.
- Cut, Copy, Paste, and Delete.
- Select All.
- Find/Replace.

It also provides these features:

- Enable Line Numbers.
- Highlight Current Line.
- Word Wrap — lines longer than the number of characters that can be displayed are wrapped in the display area so that you do not have to scroll horizontally.
- Show 80 Column Guide.
- Go to Line.

For more information on what the different parts of the protocol description file are for, see:

- Chapter 2, “Creating a New Protocol Description,” starting on page 17
- Chapter 6, “XML Element Reference,” starting on page 71

## Checking Protocol Description File Edits

**Validation Checks**

Validation checks occur as you edit a protocol description file, and messages appear in the lower portion of the editor window to tell you about problems. Validation checks find problems like:

- Incorrect element and attribute names and capitalization.
- Mismatched start and end tags.
- Invalid attribute values.
- Out of place elements.

The validation checks help you fix protocol description file problems before the file is loaded at application startup.

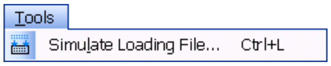
**Parsing Checks at Application Startup**

When the *Keysight Logic Analyzer* application starts up, or when you refresh protocol files in the Packet Decoder tool, protocol description files are parsed and loaded. Additional checks are performed on the files as they are parsed.

The protocol description file editor lets you simulate parsing, so that you can find additional problems before trying to use a description.

To simulate the parsing that happens at application startup or refresh:

1 From the Protocol Description File Editor's main menu, choose **Tools>Simulate Loading File**.



Messages from the parse operation appear in a separate dialog — similar to what is shown when errors are present at application startup or refresh.

## Saving Protocol Description Files

Protocol description files are located in the Protocols folder in the Keysight Logic Analyzer application's install directory. For example, the default location is: C:\Program Files\Keysight Technologies\Logic Analyzer\Protocols.

1. From the Protocol Description File Editor's main menu, choose **File>Save As...**.
2. In the Save As dialog, enter the protocol description file name; then, click **Save**.

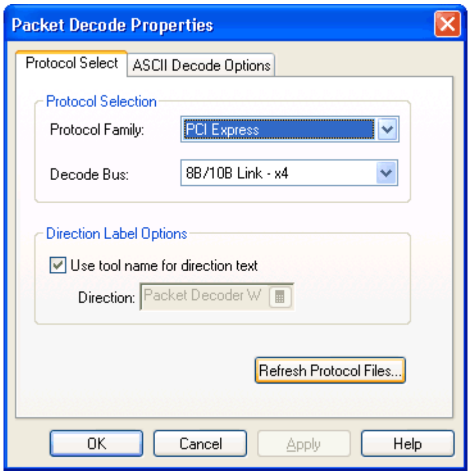
   Protocol description files have the .aex (Keysight Encrypted XML) file extension.

## Refreshing Protocol Files in the Application

In order for the *Keysight Logic Analyzer* application to recognize protocol description file changes, you must either refresh protocol files in the Packet Decoder tool or restart the *Keysight Logic Analyzer* application.

### To refresh protocol files

1. In the Overview window of the *Keysight Logic Analyzer* application, click **Properties...** in your Packet Decoder tool.
2. In the Packet Decode Properties dialog, click **Refresh Protocol Files...**.



Protocol Family: PCI Express

Decode Bus: 8B/10B Link - x4

☑ Use tool name for direction text

Direction: Packet Decoder W

3. In the Question dialog that appears, click **Yes** to save the current configuration. The saved configuration will automatically be re-opened after the protocol description files are refreshed.

   If you click **No**, the protocol files are still refreshed; however, you must either open a different configuration or create a new configuration.

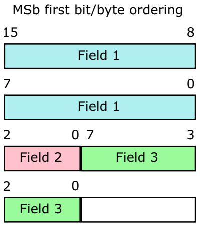
# 2 Creating a New Protocol Description

# Before You Get Started

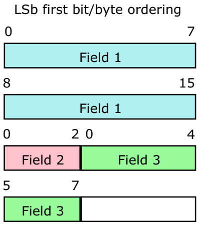
- "Byte/Bit Order Requirements" on page 18

## Byte/Bit Order Requirements

The Protocol Decode tool assumes a most-significant bit (MSb) first ordering in the packet data.

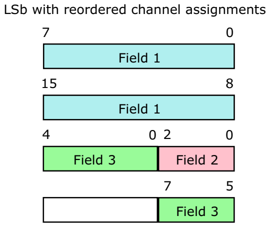


If your protocol uses a least-significant bit (LSb) first ordering in the packet data:



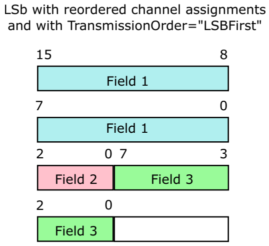
In this case, you must:

1. Reorder the bits of buses when defining the bus channel assignments (see "To assign channels, selecting the bit order" (in the online help)).



   This reverses the bit order and makes transmission order rearrangement faster.

2. Use the "TransmissionOrder='LSBFirst'" attribute within the <ProtocolFamily> element.


LSb with reordered channel assignments
and with TransmissionOrder="LSBFirst"
15
8
Field 1
7
0
Field 1
2
0
7
3
Field 2
Field 3
2
0
Field 3

This causes fields and bits within fields to be rearranged according to the field descriptions.

## Getting Started, Using a Simple Example

- "Step 1: Open the protocol description editor" on page 20
- "Step 2: Start with a minimal protocol description" on page 20
- "Step 3: Save the description to the Protocols directory" on page 22
- "Step 4: Look at results in the user interface" on page 22

### Step 1: Open the protocol description editor

See "Starting the Protocol Description File Editor" on page 10.

### Step 2: Start with a minimal protocol description

A minimal protocol description file looks something like:

```
<!-- This is a default protocol file created for you.  It is a
     starting point for the creation of a new protocol file.  Please
     refer to the online help for additional assistance. -->

<ProtocolFamily Name="Default Protocol" Version="1.1">

  <!-- Specifies a bus that encompasses a grouping of bus/signals for
       decode -->
  <Bus Name="My Bus Name">
    <!-- 'My Bus 1' and 'My Bus 2' are required bus/signals -->
    <Label Name="My Bus 1" Width="1" Type="Frame"/>
    <Label Name="My Bus 2" Width="8" Type="Data"
        Sop="'My Bus 1'==#h1"/>

    <!-- Once packets have been found, use the following protocol name
         to decode them. -->
    <BusProtocol Name="Header Protocol" Type="Packet"/>
  </Bus>

  <!-- The <PacketTypes> section defines the different types of
       packets for this protocol.  The types here are used by the
       decoder to colorize the packet types and also display the
       packet type name in the packet viewer.  The Event Editor uses
       this section to populate the list of predefined packets for
       triggering, searching, and filtering. -->
  <PacketTypes Name="Default Protocol" Protocol="Header Protocol">
    <PacketType Name="Read Packet">
      <PacketMask Width="2" Value="#h0"/>
      <PacketDisplay BackgroundColor="LightBlue"/>
    </PacketType>
    <PacketType Name="Write Packet">
      <PacketMask Width="2" Value="#h1"/>
      <PacketDisplay BackgroundColor="Yellow"/>
    </PacketType>
  </PacketTypes>

  <!-- This section describes how packets of data are decoded. -->
  <Protocol Name="Header Protocol" ProtocolLayer="Physical Layer">
    <Header>
      <Field Name="Packet Type" Length="2" Enumset="PacketTypes"/>
      <Field Name="Address" Length="4"/>
      <Field Name="Length" Length="8" Select="'Packet Type'==#h1"/>
    </Header>
    <Payload>
      <Field Name="Payload" Type="Payload" Select="'Packet Type'==#h1"
          Length="Length*8"/>
    </Payload>
  </Protocol>

  <Enumset Name="PacketTypes">
    <Enum Name="Read" Value="0"/>
    <Enum Name="Write" Value="1"/>
    <Default Name="Error" ValueError="Unknown Packet Type"/>
  </Enumset>

  <!-- This section describes how the packet viewer should be
       displayed by default. -->
  <DisplayDefaults>
    <DisplayField Name="Sample Number" Width="20"/>
    <DisplayField Name="Default Protocol Packet" Width="30"/>
    <DisplayField Name="Address" Width="20"/>
    <DisplayField Name="Length" Width="15"/>
    <DisplayField Name="Time"/>
  </DisplayDefaults>

  <!-- This section describes the errors that are possible with this
       protocol. -->
  <!-- Protocol errors are always displayed in red. -->
  <ProtocolErrors>
    <ProtocolError Name="Unknown Packet Type"
        Description="There was an undefined packet type."/>
  </ProtocolErrors>

</ProtocolFamily>
```

This is the simple protocol description that appears when you first open the protocol description file editor.

The <ProtocolFamily> element identifies the family of protocols described in the file. Within the <ProtocolFamily> element are:

- The <Bus> element is used to identify the probed buses/signals, their contents, and the packet framing.
- The <PacketTypes> element is used to identify the main packet types that will appear in the Packet Viewer's main protocol decode column and to group packet descriptions to make setting up packet triggers easier.
- The <Protocol> element defines the header and payload fields used when decoding the data. Fields are described in the same order as they appear in the packet data.
- The <DisplayDefaults> element identifies the fields that are displayed by default in the Packet Viewer window. You can always add or delete field columns from within the Packet Viewer window.

### Step 3: Save the description to the Protocols directory

Protocol description files are located in the Protocols folder in the Keysight Logic Analyzer application's install directory. For example, the default location is: C:\Program Files\Keysight Technologies\Logic Analyzer\Protocols.

1. From the Protocol Description File Editor's main menu, choose **File>Save As...**.
2. In the Save As dialog, enter the protocol description file name; then, click **Save**.

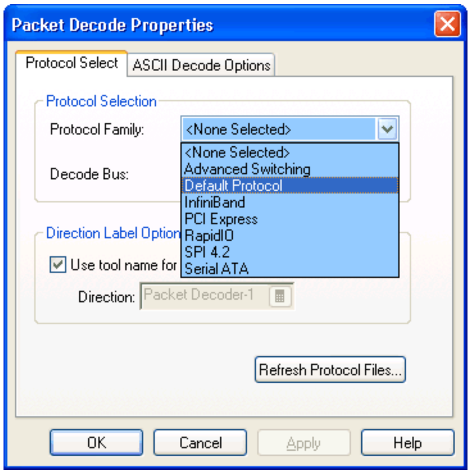
   Protocol description files have the .aex (Keysight Encrypted XML) file extension.

### Step 4: Look at results in the user interface

As you develop your protocol description, you will want to iteratively look at results in the *Keysight Logic Analyzer* application, first to see if the data is framed correctly, then to look at the decode and display results.

1. Start the *Keysight Logic Analyzer* application, or refresh the protocol files in the Packet Decoder tool (see "Refreshing Protocol Files in the Application" on page 16).

   If you see protocol description file errors when starting the application, see "Protocol Description Errors when Application Starts" on page 62.
2. Capture, open, or import the data.
3. Add a Packet Decoder tool:
   - a Choose **Tools>New Packet Decoder...**
   - b In the Protocol Select tab of the Packet Decode Properties dialog, select your **Protocol Family**.

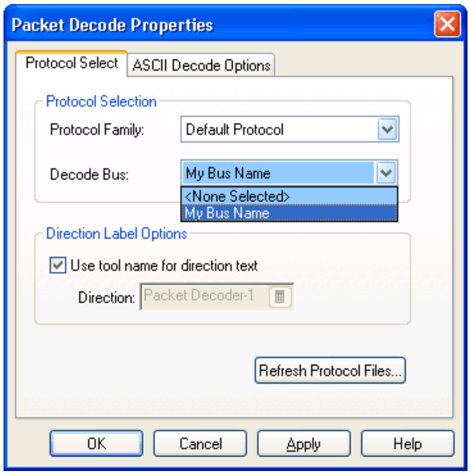


Protocol Family: <None Selected>

☑ Use tool name for

Direction: Packet Decoder-1

   - c Next, select your **Decode Bus**.



Protocol Family: Default Protocol

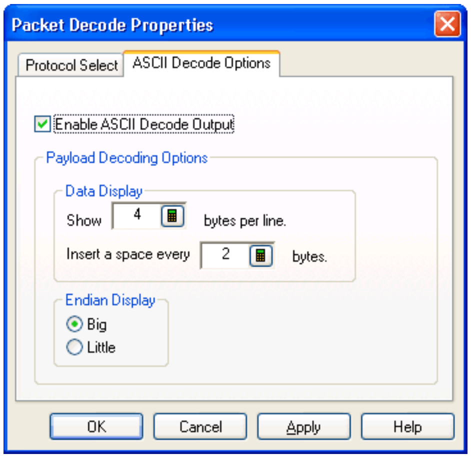
Decode Bus: My Bus Name

<None Selected>
My Bus Name

☑ Use tool name for direction text

Direction: Packet Decoder-1

d In the ASCII Decode Options tab, check the **Enable ASCII Decode Output** option.



☑ Enable ASCII Decode Output

Show 4 bytes per line.

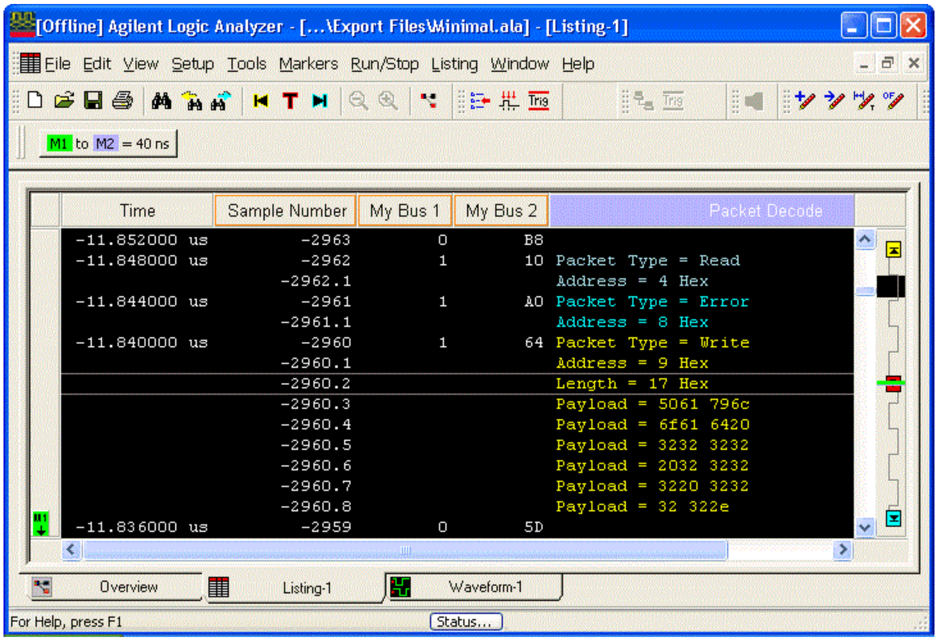
Insert a space every 2 bytes.

● Big
○ Little

A dialog informs you about the effect this option has on performance, but when developing protocol descriptions, it is sometimes useful to see packet decode information in the Listing window.

e Click **OK** to close the Packet Decode Properties dialog.

4 Look at the Packet Decode column in the Listing window.

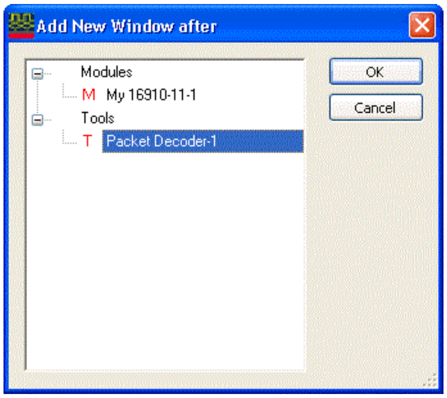

[Offline] Agilent Logic Analyzer - [...\Export Files\Minimal.ala] - [Listing-1]
File Edit View Setup Tools Markers Run/Stop Listing Window Help
M1 to M2 = 40 ns
Overview
Listing-1
Waveform-1
For Help, press F1
Status...

```
        Time          Sample Number   My Bus 1   My Bus 2   Packet Decode
  -11.852000 us              -2963          0         B8
  -11.848000 us              -2962          1         10   Packet Type = Read
                           -2962.1                          Address = 4 Hex
  -11.844000 us              -2961          1         A0   Packet Type = Error
                           -2961.1                          Address = 8 Hex
  -11.840000 us              -2960          1         64   Packet Type = Write
                           -2960.1                          Address = 9 Hex
                           -2960.2                          Length = 17 Hex
                           -2960.3                          Payload = 5061 796c
                           -2960.4                          Payload = 6f61 6420
                           -2960.5                          Payload = 3232 3232
                           -2960.6                          Payload = 2032 3232
                           -2960.7                          Payload = 3220 3232
                           -2960.8                          Payload = 32 322e
  -11.836000 us              -2959          0         5D
```

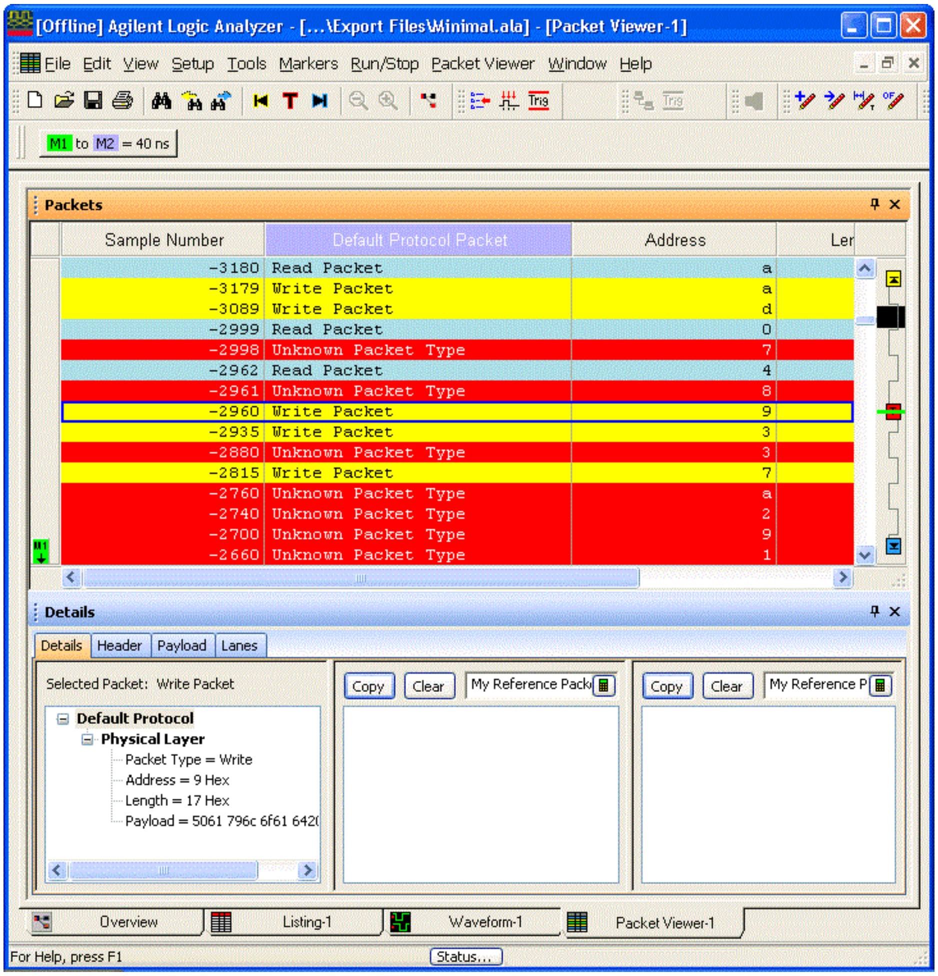
5 Add a Packet Viewer window:

*a* Choose **Window>New Packet Viewer...**

*b* In the "Add New Window after" dialog, make sure you add the Packet Viewer window after the Packet Decoder tool.



*c* Click **OK**.



| Sample Number | Default Protocol Packet | Address | Ler |
|---|---|---|---|
| -3180 | Read Packet | a | |
| -3179 | Write Packet | a | |
| -3089 | Write Packet | d | |
| -2999 | Read Packet | 0 | |
| -2998 | Unknown Packet Type | 7 | |
| -2962 | Read Packet | 4 | |
| -2961 | Unknown Packet Type | 8 | |
| -2960 | Write Packet | 9 | |
| -2935 | Write Packet | 3 | |
| -2880 | Unknown Packet Type | 3 | |
| -2815 | Write Packet | 7 | |
| -2760 | Unknown Packet Type | a | |
| -2740 | Unknown Packet Type | 2 | |
| -2700 | Unknown Packet Type | 9 | |
| -2660 | Unknown Packet Type | 1 | |

Selected Packet: Write Packet

- **Default Protocol**
  - **Physical Layer**
    - Packet Type = Write
    - Address = 9 Hex
    - Length = 17 Hex
    - Payload = 5061 796c 6f61 642

If you see protocol errors in the Packet Viewer window, see "Pre-Defined Protocol Errors that Appear in Packet Viewer" on page 66.

The results of this simple example (and its data) appear in the configuration file: C:\Documents and Settings\All Users\Documents\Keysight Technologies\Logic Analyzer\Default Configs\Keysight\Protocol Development Kit (PDK) Demo Config Files\DefaultProtocol.xml

## Getting Started, Describing Your Protocol

- "Step 5: Choose a unique protocol family name" on page 26
- "Step 6: Describe the bus to be decoded" on page 27
- "Step 7: Describe the packet types" on page 33
- "Step 8: Describe the protocol's headers, data, trailers, and fields" on page 35
- "Step 9: Describe the columns displayed in Packet Viewer by default" on page 41
- "Getting Started Summary" on page 42

### Step 5: Choose a unique protocol family name

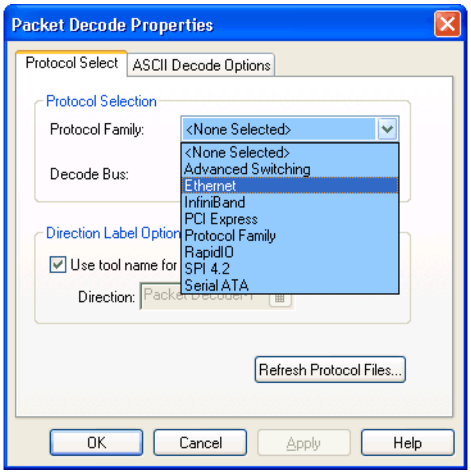
The first thing to do when working on a protocol description (for your data) is to name the <ProtocolFamily>:

- Make sure the protocol family name is unique; otherwise, the protocol descriptions may not be read by the Packet Decoder tool.
- Make sure the Version attribute is specified (must be equal to "1.1" for *Keysight Logic Analyzer* application version 03.65).
- Make sure you change the generated field name for the protocol family within the <DisplayDefaults> element.

For example:

```
<ProtocolFamily Name="Ethernet" Version="1.1">
   ...

   <DisplayDefaults>
      ...
      <DisplayField Name="Ethernet Packet" Width="25"/>
      ...
   </DisplayDefaults>

</ProtocolFamily>
```

The name you give a protocol family appears as a choice when selecting the properties of a Packet Decoder tool.



### Step 6: Describe the bus to be decoded

After choosing a protocol family name, you need to describe the bus to be decoded. This is done with the <Bus> element and its child elements. Use the <Bus> element's Name attribute to name the bus.

Next, you need to describe the buses/signals that will be used for decoding with <Label> elements inside the <Bus> element.

Buses/signals can be from logic analyzer modules (that probe a device under test), from imported data, or from upstream tools that process captured or imported data in some way. A <Label> element's Name attribute must be a bus/signal name defined in a logic analyzer module (see "Defining Buses and Signals" (in the online help)), data import module, or generated by an upstream tool.

Buses/signals used for decoding can contain the following types of information:

- Data — that is, the actual data to be decoded.
- Framing — signals that identify start-of-packet and end-of-packet.
- Validity — signals that identify when data is valid.
- 8B/10B — signals that identify a switch between 8B and 10B data or identify K-characters.

The data to be decoded comes from <Label> elements whose Type attribute identifies data (see “Labels that Contain Data" on page 28). The Sop="(formula)" attribute, the Eop="(formula)" attribute (if used), and the Valid="(formula)" attribute (if used) can be used with any <Label> element whose Type attribute identifies data. The formula values of the Sop, Eop, and Valid attributes make reference to other <Label> elements that have framing or validity information.

**NOTE** **The Sop, Eop, and Valid attributes are used in the <Label> element that identifies the data (not the in the <Label> elements for signals that define the start-of-packet, end-of-packet, or when data is valid).**

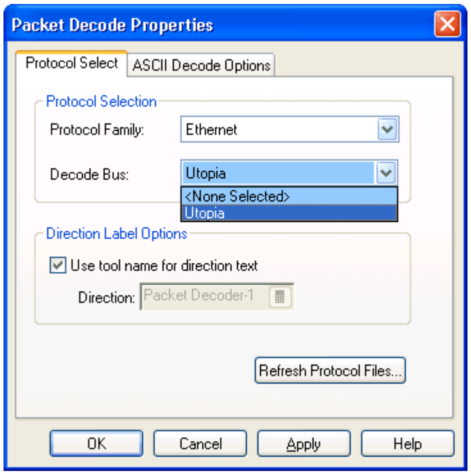
The last part of describing buses/signals is identifying the protocol used to decode the bus; this is done with the <BusProtocol> element. Note that the <BusProtocol> name you specify must match a defined <Protocol> description name. (The <PacketTypes> element's Protocol attribute must also match a defined <Protocol> description name.)

For example, suppose the bus you are probing is 16 bits wide. Additionally, there is one signal that specifies when a start-of-packet is present on the bus. Also, an end-of-packet signal is probed to specify when a packet ends. There is a signal named "CLK" that specifies when valid data is on the "DATA" bus. When CLK is 0, data will be completely ignored, and SOP/EOP will not be considered.

```
<ProtocolFamily Name="Ethernet" Version="1.1">

   <Bus Name="Utopia">
      <Label Name="DATA" Width="16" Type="Data" Sop="'SOP'==#b1"
            Eop="'EOP'==#b1" Valid="'CLK'==#b1" />
      <Label Name="SOP" Width="1" Type="Frame"/>
      <Label Name="EOP" Width="1" Type="Frame"/>
      <Label Name="CLK" Width="1" Type="Valid"/>
      <BusProtocol Name="IEEE 802.3 (Ethernet V2)" Type="Packet"/>
   </Bus>

   ...

   <Protocol Name="IEEE 802.3 (Ethernet V2)" ... >
      ...
   </Protocol>
```

```
    ...

    <PacketTypes ... Protocol="IEEE 802.3 (Ethernet V2)" ... >
       ...
    </PacketTypes>

</ProtocolFamily>
```

The bus names you describe appear as choices when selecting the properties of a Packet Decoder tool or when setting up a packet trigger.



Protocol Family: Ethernet

Decode Bus: Utopia

<None Selected>
Utopia

☑ Use tool name for direction text

Direction: Packet Decoder-1

**See Also**

- "Labels that Contain Data" on page 28
- "Labels that Identify Valid Data" on page 29
- "If Your Serial Bus Has Lanes" on page 30

If your serial bus does not have lanes, you can choose not to display the Lanes tab in the Packet Viewer window (right-click in the lower pane and deselect **Display>Lanes**).

### Labels that Contain Data

Data can be decoded from <Label> elements that have the Type="Data", Type="8bData", Type="10bData", or Type="MetaData" attribute.

The Type="8bData" and Type="10bData" attributes simply describe data values that are 8b or 10b values.

The Type="MetaData" attribute provides a convenient way to partition a label's data into additional generated labels. For example:

```
<Bus Name="PIPE 16-bit - x2" Style="Serial" LogicalLanes="2"
      PhysicalLanes="4" ProtocolBits="32">
   <Label Name="TxData0" Width="16" Type="MetaData"/>
   <Label Name="TxData1" Width="16" Type="MetaData"/>
   <Label Name="Lane0" Width="8" Type="8bData" Lane="0"
         Value="TxData0[7:0]"  Kchar="TxK0[0]==#h1"
         OrderedSetSop="Lane0==#hbc .land. TxK0[0]==#h1"
         Sop="(Lane0==#hfb || Lane0==#h5c) .land. TxK0[0]==#h1"
         Eop="(Lane0==#hfd || Lane0==#hfe) .land. TxK0[0]==#h1" />
   <Label Name="Lane1" Width="8" Type="8bData" Lane="1"
```

```
            Value="TxData1[7:0]"  Kchar="TxK1[0]==#h1"
            OrderedSetSop="Lane1==#hbc .land. TxK1[0]==#h1"
            Sop="(Lane1==#hfb || Lane1==#h5c) .land. TxK1[0]==#h1"
            Eop="(Lane1==#hfd || Lane1==#hfe) .land. TxK1[0]==#h1" />
      <Label Name="Lane2" Width="8" Type="8bData" Lane="2"
            Value="TxData0[15:8]" Kchar="TxK0[1]==#h1"
            OrderedSetSop="Lane2==#hbc .land. TxK0[1]==#h1"
            Sop="(Lane2==#hfb || Lane2==#h5c) .land. TxK0[1]==#h1"
            Eop="(Lane2==#hfd || Lane2==#hfe) .land. TxK0[1]==#h1" />
      <Label Name="Lane3" Width="8" Type="8bData" Lane="3"
            Value="TxData1[15:8]" Kchar="TxK1[1]==#h1"
            OrderedSetSop="Lane3==#hbc .land. TxK1[1]==#h1"
            Sop="(Lane3==#hfb || Lane3==#h5c) .land. TxK1[1]==#h1"
            Eop="(Lane3==#hfd || Lane3==#hfe) .land. TxK1[1]==#h1" />
      <Label Name="TxK0" Width="2" Type="K/D"/>
      <Label Name="TxK1" Width="2" Type="K/D"/>
      <BusProtocol Name="PCI Express Packet" Type="Packet"/>
      <BusProtocol Name="PCI Express Lane" Type="Lane"/>
      <BusProtocol Name="PCI Express Symbol" Type="Symbol"/>
   </Bus>
```

When there are multiple <Label> elements with Type attributes that identify data, the Select attribute is used to identify which label to get data from.

### Labels that Identify Valid Data

Valid data is identified by <Label> elements with the Type="Valid", Type="Bonded", or Type="Idle" attributes. The PacketData attribute provides another level of identifying valid data. Invalid data is not decoded.

The Type="Valid" attribute identifies a label that specifies when data is valid. This attribute is usually used for single lanes of data. Labels with Type="Valid" are used in conjunction with the Valid attribute in the label whose type identifies data. A Valid attribute formula that results in "1" means the data is valid, and a formula that results in "0" means the data is invalid. For example:

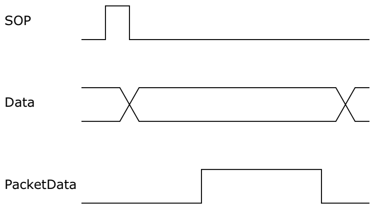
```
<Bus ... >
   <Label Name="My Bus 1" Type="Data" ...
         Valid="'My Bus 2'==#h7" />
   <Label Name="My Bus 2" Width="3" Type="Valid"/>
```

The Type="Bonded" attribute specifies when multi-lane data is bonded (or aligned). This is another way of identifying valid data. A value of "1" means the data is valid, and "0" means the data is invalid.

The Type="Idle" attribute specifies when data is invalid due to being idle. A value of "0" means the data is valid, and "1" means the data is invalid.

Labels with Type="Bonded" or Type="Idle" identify signals that automatically filter samples. They are not like labels with Type="Valid" that are used in conjunction with attributes in the data-type label.

The PacketData="(formula)" attribute provides another level of identifying when data is valid. For example, the PacketData formula can identify subsections of data that contain valid packet data:



For example:

```
<Bus Name="LPC" ProtocolBits="8">
   <Label Name="LFRAME" Width="1" Type="Frame" />
   <Label Name="Cycle Type" Width="32" Type="Valid" />
   <Label Name="LAD" Width="4" Type="Data" Lane="0"
         Sop="LFRAME==0 .land. LFRAME{1}==1"
         PacketData="'Cycle Type'!=#h21 .land.
                     'Cycle Type'!=#h401 .land.
                     'Cycle Type'!=#ha01" />
   <BusProtocol Name="LPC Packet" Type="Packet" />
</Bus>
```

**NOTE** **PacketData attribute formula strings must appear on one line. The formatting in the example above is for readability.**

---

### If Your Serial Bus Has Lanes

If your serial bus has *lanes* (see page 113) (see also Chapter 5, “Multi-Lane Serial Link Concepts,” starting on page 67):

- You will have <Label> elements with Sop and Lane attributes for each physical lane.
- <BusProtocol> elements can be used for packet, lane (ordered set), and symbol decoding, and there will be corresponding <Protocol> descriptions for packet and lane decoding and <SymbolDecode> elements for symbol decoding.
- You may want to display lane data in the Listing window.

For example, here is a bus description from the PCI Express protocol description file:

```
<ProtocolFamily Name="PCI Express" Version="1.1">

   <Bus Name="8B/10B Link - x2" GenerateLaneData="T" LogicalLanes="2"
         PhysicalLanes="4" MaxBytes="10000" ProtocolBits="32">
      <Label Name="10bbyte0" Width="10" Type="10bData" Lane="0"
            Select="'8b/10b'[3]==#h0" Valid="valid[3]==#h1"
            OrderedSetSop="'10bbyte0'==#hea"
            Sop="'10bbyte0'==#h368 ||
                 '10bbyte0'==#h97 ||
                 '10bbyte0'==#hf5 ||
                 '10bbyte0'==#h30a"
            Eop="'10bbyte0'==#h2e8 ||
                 '10bbyte0'==#h117 ||
                 '10bbyte0'==#h1e8 ||
                 '10bbyte0'==#h217" />
      <Label Name="10bbyte1" Width="10" Type="10bData" Lane="1"
            Select="'8b/10b'[2]==#h0" Valid="valid[2]==#h1"
```

```
        OrderedSetSop="'10bbyte1'==#hea"
        Sop="'10bbyte1'==#h368 ||
             '10bbyte1'==#h97 ||
             '10bbyte1'==#hf5 ||
             '10bbyte1'==#h30a"
        Eop="'10bbyte1'==#h2e8 ||
             '10bbyte1'==#h117 ||
             '10bbyte1'==#h1e8 ||
             '10bbyte1'==#h217" />
  <Label Name="10bbyte2" Width="10" Type="10bData" Lane="2"
        Select="'8b/10b'[1]==#h0" Valid="valid[1]==#h1"
        OrderedSetSop="'10bbyte2'==#hea"
        Sop="'10bbyte2'==#h368 ||
             '10bbyte2'==#h97 ||
             '10bbyte2'==#hf5 ||
             '10bbyte2'==#h30a"
        Eop="'10bbyte2'==#h2e8 ||
             '10bbyte2'==#h117 ||
             '10bbyte2'==#h1e8 ||
             '10bbyte2'==#h217" />
  <Label Name="10bbyte3" Width="10" Type="10bData" Lane="3"
        Select="'8b/10b'[0]==#h0" Valid="valid[0]==#h1"
        OrderedSetSop="'10bbyte3'==#hea"
        Sop="'10bbyte3'==#h368 ||
             '10bbyte3'==#h97 ||
             '10bbyte3'==#hf5 ||
             '10bbyte3'==#h30a"
        Eop="'10bbyte3'==#h2e8 ||
             '10bbyte3'==#h117 ||
             '10bbyte3'==#h1e8 ||
             '10bbyte3'==#h217" />
  <Label Name="8bbyte0" Width="8" Type="8bData" Lane="0"
        Select="'8b/10b'[3]==#h1" Valid="valid[3]==#h1"
        Kchar="kcode[3]==#h1"
        Sop="('8bbyte0'==#hfb || '8bbyte0'==#h5c) .land.
             kcode[3]==#h1"
        Eop="('8bbyte0'==#hfd || '8bbyte0'==#hfe) .land.
             kcode[3]==#h1" />
  <Label Name="8bbyte1" Width="8" Type="8bData" Lane="1"
        Select="'8b/10b'[2]==#h1" Valid="valid[2]==#h1"
        Kchar="kcode[2]==#h1"
        Sop="('8bbyte1'==#hfb || '8bbyte1'==#h5c) .land.
             kcode[2]==#h1"
        Eop="('8bbyte1'==#hfd || '8bbyte1'==#hfe) .land.
             kcode[2]==#h1" />
  <Label Name="8bbyte2" Width="8" Type="8bData" Lane="2"
        Select="'8b/10b'[1]==#h1" Valid="valid[1]==#h1"
        Kchar="kcode[1]==#h1"
        Sop="('8bbyte2'==#hfb || '8bbyte2'==#h5c) .land.
             kcode[1]==#h1"
        Eop="('8bbyte2'==#hfd || '8bbyte2'==#hfe) .land.
             kcode[1]==#h1" />
  <Label Name="8bbyte3" Width="8" Type="8bData" Lane="3"
        Select="'8b/10b'[0]==#h1" Valid="valid[0]==#h1"
        Kchar="kcode[0]==#h1"
        Sop="('8bbyte3'==#hfb || '8bbyte3'==#h5c) .land.
             kcode[0]==#h1"
```

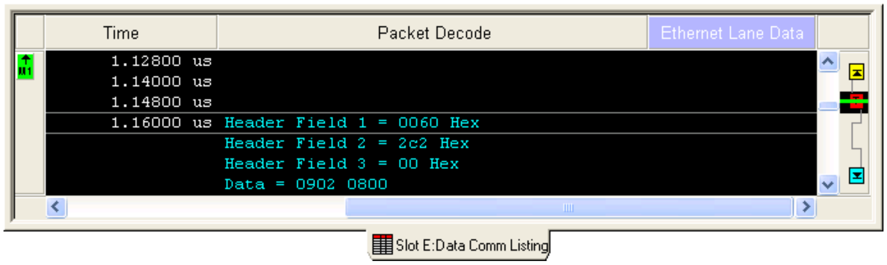
```
            Eop="('8bbyte3'==#hfd || '8bbyte3'==#hfe) .land.
                 kcode[0]==#h1" />
      <Label Name="valid" Width="4" Type="Valid"/>
      <Label Name="8b/10b" Width="4" Type="8b/10b"/>
      <Label Name="kcode" Width="4" Type="K/D"/>
      <Label Name="bonded" Width="1" Type="Bonded"/>
      <Label Name="linkidle" Width="1" Type="Idle"/>
      <BusProtocol Name="PCI Express Packet" Type="Packet"/>
      <BusProtocol Name="PCI Express Lane" Type="Lane"/>
      <BusProtocol Name="PCI Express Symbol" Type="Symbol"/>
   </Bus>

   ...

   <Protocol Name="PCI Express Packet" ProtocolLayer="Physical">
      ...
   </Protocol>

   <Protocol Name="PCI Express Lane" Type="Lane"
         ProtocolLayer="Physical">
      ...
   </Protocol>

   <SymbolDecode>
      <Enumset Name="PCI Express Symbol">
         ...
      </Enumset>
   </SymbolDecode>


   ...

   <PacketTypes Name="PCI Express" Protocol="PCI Express Packet">
      ...
   </PacketTypes>

</ProtocolFamily>
```

**NOTE** **The Sop and Eop attribute formula strings must appear on one line. The formatting in the example above is for readability.**

---

## Generating Lane Data in the Listing Window

You can use the <Bus> element's GenerateLaneData attribute to generate lane data in the Listing window. When GenerateLaneData="T", a generated bus/signal column named "(ProtocolFamily name) Lane Data" is added to the Listing window (when ASCII decode output is enabled). For example, if your protocol family name is "Ethernet", a column named "Ethernet Lane Data" can appear in the Listing window.



```
1.12800 us
1.14000 us
1.14800 us
1.16000 us Header Field 1 = 0060 Hex
           Header Field 2 = 2c2 Hex
           Header Field 3 = 00 Hex
           Data = 0902 0800
```

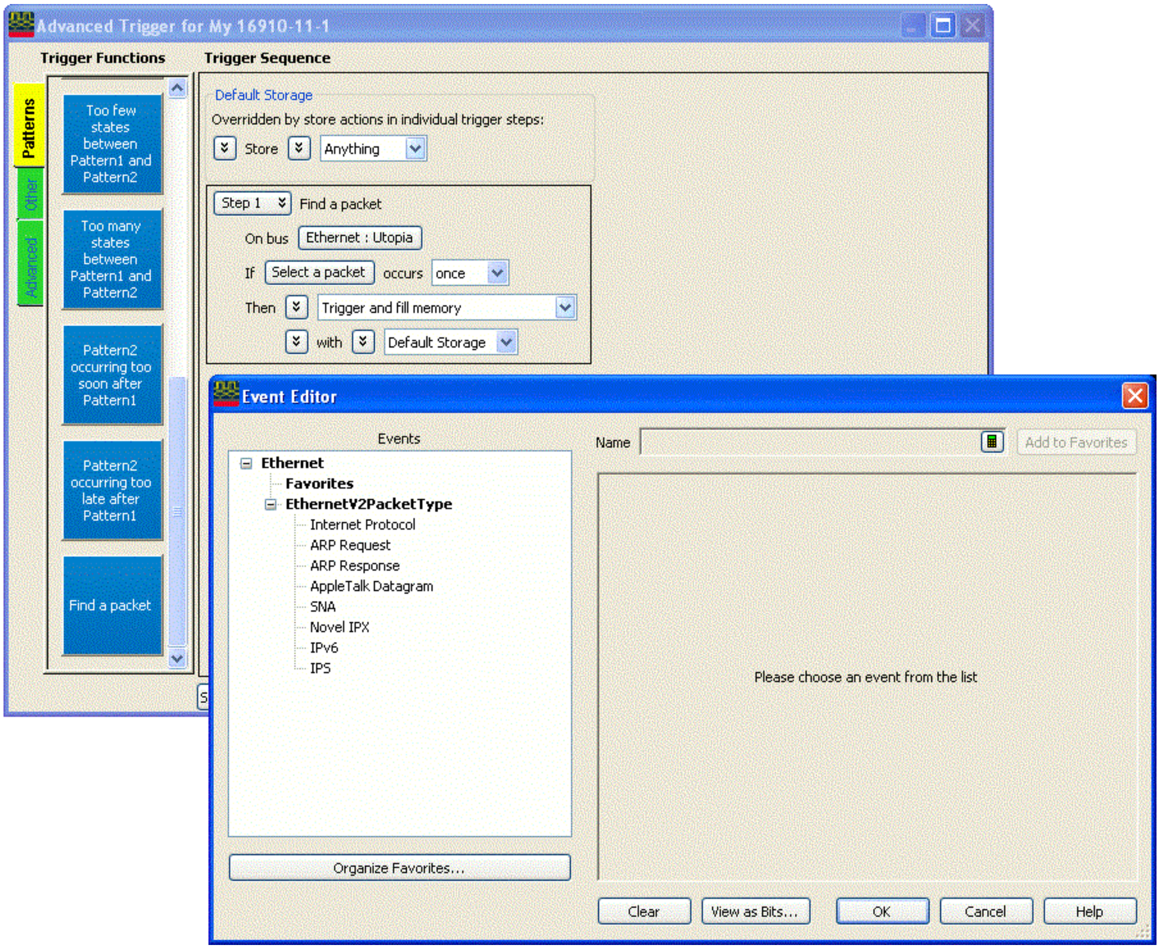
### Step 7: Describe the packet types

After describing the bus to be decoded, describe the packet types that will appear in the Packet Viewer's decode column (and in the dialogs used for setting up packet triggers) by using the <PacketTypes>, <PacketTypeGroup>, and <PacketType> elements.

```
<ProtocolFamily Name="Ethernet" Version="1.1">

   <PacketTypes Name="EthernetV2PacketType"
         Protocol="IEEE 802.3 (Ethernet V2)"
         Default="Internet Protocol">
      <PacketTypeGroup Name="EthernetV2PacketType">
         <PacketType Name="Internet Protocol">
            <PacketMask BitOffset="96" Width="16" Value="#h0800"/>
         </PacketType>
         <PacketType Name="ARP Request">
            <PacketMask BitOffset="96" Width="16" Value="#h0806"/>
         </PacketType>
         <PacketType Name="ARP Response">
            <PacketMask BitOffset="96" Width="16" Value="#h0835"/>
         </PacketType>
         <PacketType Name="AppleTalk Datagram">
            <PacketMask BitOffset="96" Width="16" Value="#h809b"/>
         </PacketType>
         <PacketType Name="SNA">
            <PacketMask BitOffset="96" Width="16" Value="#h80d5"/>
         </PacketType>
         <PacketType Name="Novel IPX">
            <PacketMask BitOffset="96" Width="16" Value="#h8137"/>
         </PacketType>
         <PacketType Name="IPv6">
            <PacketMask BitOffset="96" Width="16" Value="#h86dd"/>
         </PacketType>
         <PacketType Name="IPS">
            <PacketMask BitOffset="96" Width="16" Value="#h2007"/>
         </PacketType>
      </PacketTypeGroup>
   </PacketTypes>

</ProtocolFamily>
```

In the <PacketTypes> element, the Default="(packet type name)" attribute specifies the default packet type in the Event Editor.

You can use additional <PacketTypeGroup> elements to organize a hierarchy of packet types that can be selected from when setting up packet triggers.



Overridden by store actions in individual trigger steps:

Store Anything

Step 1 Find a packet

On bus Ethernet : Utopia

If Select a packet occurs once

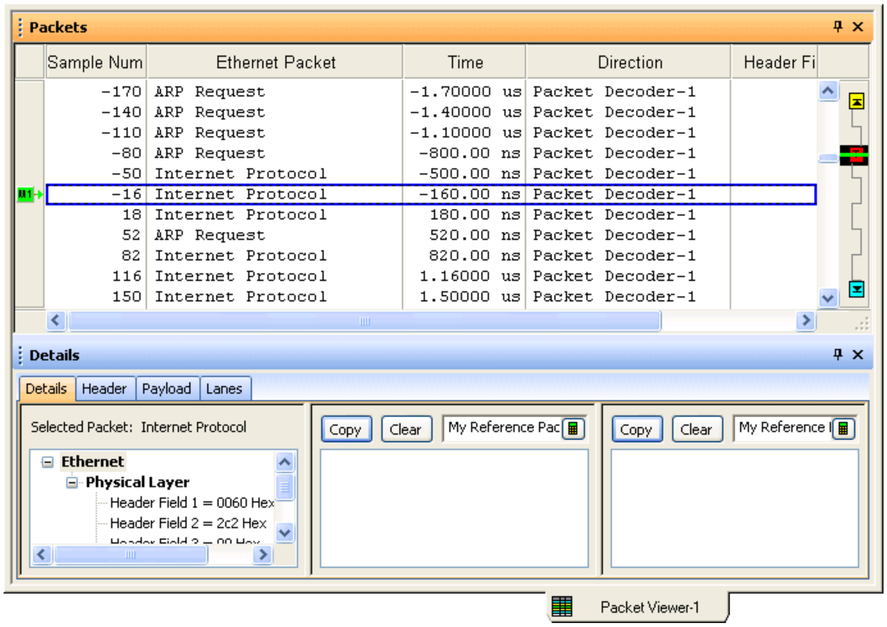
Then Trigger and fill memory

with Default Storage

Please choose an event from the list

The <PacketMask> element identifies the specific bit values within the packet that identify the packet type. If the bits that specify the packet type are not the first bits in the packet, you can use the BitOffset="(bit offset)" attribute to specify where those bits are located.

The described packet types appear in the main packet decode column of the Packet Viewer window.



| Sample Num | Ethernet Packet | Time | Direction | Header Fi |
|---|---|---|---|---|
| -170 | ARP Request | -1.70000 us | Packet Decoder-1 | |
| -140 | ARP Request | -1.40000 us | Packet Decoder-1 | |
| -110 | ARP Request | -1.10000 us | Packet Decoder-1 | |
| -80 | ARP Request | -800.00 ns | Packet Decoder-1 | |
| -50 | Internet Protocol | -500.00 ns | Packet Decoder-1 | |
| -16 | Internet Protocol | -160.00 ns | Packet Decoder-1 | |
| 18 | Internet Protocol | 180.00 ns | Packet Decoder-1 | |
| 52 | ARP Request | 520.00 ns | Packet Decoder-1 | |
| 82 | Internet Protocol | 820.00 ns | Packet Decoder-1 | |
| 116 | Internet Protocol | 1.16000 us | Packet Decoder-1 | |
| 150 | Internet Protocol | 1.50000 us | Packet Decoder-1 | |

Selected Packet: Internet Protocol

- Ethernet
  - Physical Layer
    - Header Field 1 = 0060 Hex
    - Header Field 2 = 2c2 Hex

You can color-code packet types in the Packet Viewer window and add descriptive tool tips with <PacketDisplay> elements inside the <PacketType> elements (see "Describing Packet Type Colors" on page 44).

## Step 8: Describe the protocol's headers, data, trailers, and fields

After describing packet types, start identifying protocols, their headers, payloads, and trailers, and the fields that appear in each.

Use the <Protocol> element to name the protocol description (with the Name attribute) and identify its layer within the protocol stack (with the ProtocolLayer attribute).

Use <Header>, <Payload>, and <Trailer> elements to describe these parts of a packet. Use <Field> elements within <Header>, <Payload>, and <Trailer> elements to describe fields.

For example:

```
<ProtocolFamily Name="Ethernet" Version="1.1">
   <Bus ... > ... </Bus>

   <Protocol Name="IEEE 802.3 (Ethernet V2)"
         ProtocolLayer="Physical Layer">
      <Header>
         <Field Name="Dest Addr" Length="48" Type="Data"
               Format="Hex"/>
         <Field Name="Src  Addr" Length="48" Type="Data"
               Format="Hex"/>
         <Field Name="Length/Type" Length="16"
               Enumset="EthernetV2PacketType"/>
         <Field Name="Length/Type" Type="ProtocolField"/>
      </Header>
   </Protocol>
```

In the example above, bits 0-47 of the packet are the Dest Addr, bits 48-95 are the Src Addr, and bits 96-111 are the Length/Type. The order of <Field> elements must be the same as the order of fields in the packet. The bits of the packet are counted as Length attributes are used.

The default <Field> element type is data, so the Type="Data" attribute is not necessary; however, you may want to use it in order to distinguish normal data fields from other type fields.

The Format attribute describes the default base when the field is displayed in the Packet Viewer window or the Event Editor.

Fields with Type="Protocol" or Type="ProtocolField" reference the next layers of the protocol stack, which are described with additional <Protocol> elements.

The Type="ProtocolField" attribute (see the previous example) references multiple additional protocols whose names are defined in an enumeration set. Two <Field> elements are used: the first is for decoding the value in the field, and the second says to use additional protocol descriptions for further decoding. Continuing on with the previous example:

```
   <Enumset Name="EthernetV2PacketType">
      <Enum Value="#h0800" Name="Internet Protocol"/>
      <Enum Value="#h0806" Name="ARP Request"/>
      <Enum Value="#h0835" Name="ARP Response"/>
      <Enum Value="#h809b" Name="AppleTalk Datagram"/>
      <Enum Value="#h80d5" Name="SNA"/>
      <Enum Value="#h8137" Name="Novel IPX"/>
      <Enum Value="#h86dd" Name="IPv6"/>
      <Enum Value="#h2007" Name="IPS"/>
      <Enum Value="#h6002" Name="DEC MOP Remote Console"/>
      <Enum Value="#h6004" Name="DEC LAT"/>
   </Enumset>

   <Protocol Name="Internet Protocol" ProtocolLayer="Network Layer">
   </Protocol>

   <Protocol Name="ARP Request" ProtocolLayer="Network Layer">
   </Protocol>

   <Protocol Name="ARP Response" ProtocolLayer="Network Layer">
   </Protocol>

   <Protocol Name="AppleTalk Datagram" ProtocolLayer="Network Layer">
   </Protocol>

   <Protocol Name="SNA" ProtocolLayer="Network Layer">
   </Protocol>

   <Protocol Name="Novell IPX" ProtocolLayer="Network Layer">
   </Protocol>

   <Protocol Name="IPv6" ProtocolLayer="Network Layer">
   </Protocol>

   <Protocol Name="IPS" ProtocolLayer="Network Layer">
   </Protocol>

   ...

</ProtocolFamily>
```

Enumeration sets are described with <Enumset> elements; their main purpose is to assign meaningful strings to values. For more information, see "Using Enumsets" on page 43.

You can describe the next layer of protocol by filling-in additional <Protocol> elements. For example, to describe "Internet Protocol":

```
<!-- ********************************************************** -->
<!-- IP (Internet Protocol) Packet Definition                   -->
<!-- This description assumes the packet length is 20 bytes.    -->
<!-- ********************************************************** -->

<Protocol Name="Internet Protocol" ProtocolLayer="Network Layer">
   <Header>
      <Field Name="Version" Length="4" Type="Data" Format="Hex"/>
      <Field Name="Header Length" Length="4" Type="Data"
            Format="Decimal"/>
      <Field Name="Precedence" Length="3"
            Enumset="PrecedenceSymbols"/>
      <Field Name="Delay" Length="1" Enumset="NormalLowSymbols"/>
      <Field Name="Throughput" Length="1"
            Enumset="NormalHighSymbols"/>
      <Field Name="Reliability" Length="1"
            Enumset="NormalHighSymbols"/>
      <Field Name="Cost" Length="1" Enumset="NormalLowSymbols"/>
      <Field Name="MBZ" Length="1" Type="Data" Format="Binary"/>
      <Field Name="Total Length" Length="16" Type="Data"
            Format="Decimal"/>
      <Field Name="Identification" Length="16" Type="Data"
            Format="Hex"/>
      <Field Name="Zero" Length="1" Type="Data" Format="Binary"/>
      <Field Name="Do not fragment" Length="1" Type="Data"
            Format="Binary"/>
      <Field Name="May fragment" Length="1" Type="Data"
            Format="Binary"/>
      <Field Name="Fragment Offset" Length="13" Type="Data"
            Format="Decimal"/>
      <Field Name="Time To Live" Length="8" Type="Data"
            Format="Decimal"/>
      <Field Name="Protocol" Length="8" Enumset="IPProtocolType"/>
      <Field Name="Header Checksum" Length="16" Type="Data"
            Format="Hex"/>
      <Field Name="IP Src  Addr" Length="32" Type="Data"
            Format="DotNotation"/>
      <Field Name="IP Dest Addr" Length="32" Type="Data"
            Format="DotNotation"/>
      <Field Name="Internet Cntrl Msg Protocol" Type="Protocol"
            Select="'Protocol'==#h01"/>
      <Field Name="Transmission Control Protocol" Type="Protocol"
            Select="'Protocol'==#h06"/>
      <Field Name="User Datagram Protocol" Type="Protocol"
            Select="'Protocol'==#h11"/>
      <Field Name="Open Shortest Path First IGP" Type="Protocol"
            Select="'Protocol'==#h59"/>
   </Header>
</Protocol>

<Enumset Name="PrecedenceSymbols">
```

```
      <Enum Value="#h0" Name="Routine"/>
      <Enum Value="#h1" Name="Priority"/>
      <Enum Value="#h2" Name="Immediate"/>
      <Enum Value="#h3" Name="Flash"/>
      <Enum Value="#h4" Name="Flash Override"/>
      <Enum Value="#h6" Name="Internetwork Control"/>
      <Enum Value="#h7" Name="Network Control"/>
   </Enumset>

   <Enumset Name="NormalLowSymbols">
      <Enum Value="#b0" Name="Normal"/>
      <Enum Value="#b1" Name="Low"/>
   </Enumset>

   <Enumset Name="NormalHighSymbols">
      <Enum Value="#b0" Name="Normal"/>
      <Enum Value="#b1" Name="High"/>
   </Enumset>

   <Enumset Name="IPProtocolType">
      <Enum Value="#h01" Name="Internet Cntrl Msg Protocol"/>
      <Enum Value="#h06" Name="Transmission Control Protocol"/>
      <Enum Value="#h11" Name="User Datagram Protocol"/>
      <Enum Value="#h59" Name="Open Shortest Path First IGP"/>
   </Enumset>

   <Protocol Name="Internet Cntrl Msg Protocol"
         ProtocolLayer="Transport Layer">
   </Protocol>

   <Protocol Name="Transmission Control Protocol"
         ProtocolLayer="Transport Layer">
   </Protocol>

   <Protocol Name="User Datagram Protocol"
         ProtocolLayer="Transport Layer">
   </Protocol>

   <Protocol Name="Open Shortest Path First IGP"
         ProtocolLayer="Transport Layer">
   </Protocol>
```

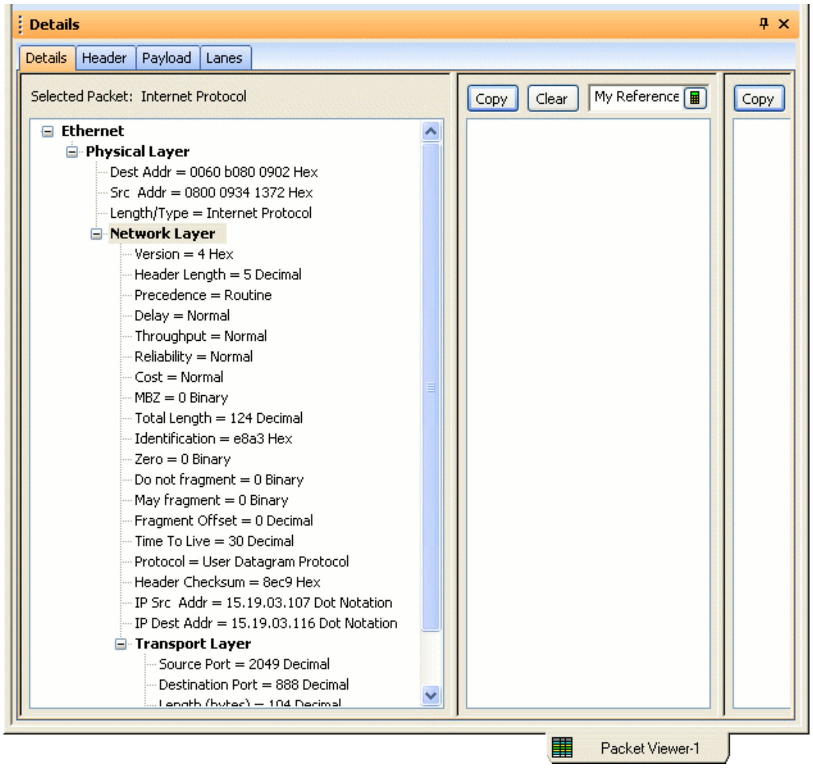
The above description shows another way to reference the next protocol level. The Type="Protocol" attribute is used within <Field> elements, and Select attributes are used to identify the protocol at the next level; in this case, the field that identifies the next level of protocol appears earlier in the packet.

Again, you can describe the next layer of protocol by filling-in additional <Protocol> elements. For example, to describe "User Datagram Protocol":

```
   <!-- ********************************************************* -->
   <!-- UPD Packet Definition                                     -->
   <!-- ********************************************************* -->

   <Protocol Name="User Datagram Protocol"
         ProtocolLayer="Transport Layer">
      <Header>
         <Field Name="Source Port" Length="16" Enumset="PortSymbols"
```

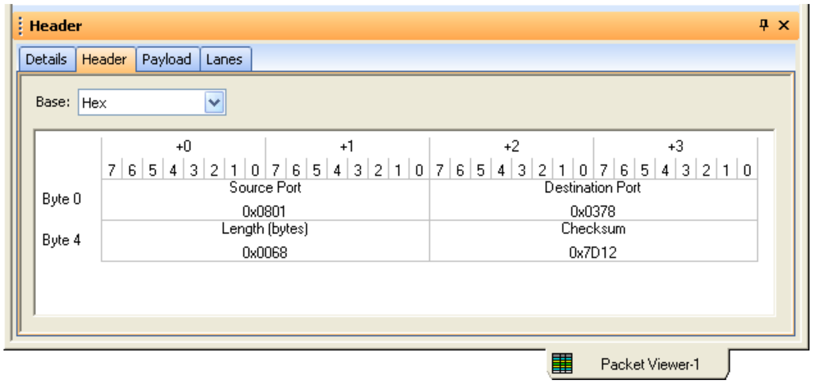
```
            Format="Decimal"/>
      <Field Name="Destination Port" Length="16"
            Enumset="PortSymbols" Format="Decimal"/>
      <Field Name="Length (bytes)" Length="16" Type="Data"
            Format="Decimal"/>
      <Field Name="Checksum" Length="16" Type="Data" Format="Hex"/>
   </Header>
   <Payload>
      <Field Name="Data" Length="'#PACKET_LENGTH' - 336"
            Type="Payload"/>
   </Payload>
</Protocol>

<Enumset Name="PortSymbols">
   <Enum Value="#d5" Name="Remote Job Entry"/>
   <Enum Value="#d7" Name="Echo"/>
   <Enum Value="#d9" Name="Discard"/>
   <Enum Value="#d11" Name="Active Users"/>
   <Enum Value="#d13" Name="Daytime"/>
   <Enum Value="#d15" Name="Who Is Up/NETSTAT"/>
   <Enum Value="#d17" Name="Quote Of The Day"/>
   <Enum Value="#d19" Name="Character Generation"/>
   <Enum Value="#d20" Name="FTP - data"/>
   <Enum Value="#d21" Name="FTP"/>
   <Enum Value="#d22" Name="SSH"/>
   <Enum Value="#d23" Name="TELNET"/>
   <Enum Value="#d25" Name="SMTP"/>
   <Enum Value="#d37" Name="Time"/>
   <Enum Value="#d39" Name="Resource Location Protocol"/>
   <Enum Value="#d42" Name="Host Name Server"/>
   <Enum Value="#d43" Name="NICNAME/Who Is"/>
   <Enum Value="#d53" Name="DNS"/>
   <Enum Value="#d67" Name="BOOTP - Server"/>
   <Enum Value="#d68" Name="BOOTP - Client"/>
   <Enum Value="#d69" Name="Trivial FTP"/>
   <Enum Value="#d75" Name="Private Dial-Out Service"/>
   <Enum Value="#d77" Name="Private RJE Service"/>
   <Enum Value="#d79" Name="Finger"/>
   <Enum Value="#d80" Name="WWW"/>
   <Enum Value="#d95" Name="SUPDUP Protocol"/>
   <Enum Value="#d101" Name="NIC Host Name Server"/>
   <Enum Value="#d102" Name="ISO-TSAP"/>
   <Enum Value="#d109" Name="POP - Post Office Prot"/>
   <Enum Value="#d110" Name="POP3 - Post Office Prot"/>
   <Enum Value="#d111" Name="Portmap"/>
   <Enum Value="#d113" Name="Authentication Service"/>
   <Enum Value="#d115" Name="SFTP"/>
   <Enum Value="#d117" Name="UUCP Path Service"/>
   <Enum Value="#d119" Name="NNTP News Transfer"/>
   <Enum Value="#d123" Name="Network Time Protocol"/>
   <Enum Value="#d137" Name="NETBIOS"/>
   <Enum Value="#d138" Name="NETBIOS"/>
   <Enum Value="#d139" Name="NETBIOS"/>
   <Enum Value="#d161" Name="SNMP"/>
</Enumset>
```

The decoded fields and values can be seen in the Packet Viewer window's Details, Header, Payload, and Lanes tabs:



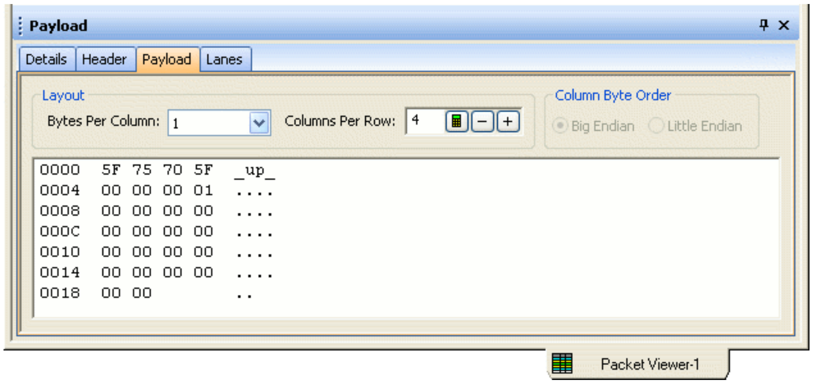
Selected Packet: Internet Protocol

- **Ethernet**
  - **Physical Layer**
    - Dest Addr = 0060 b080 0902 Hex
    - Src Addr = 0800 0934 1372 Hex
    - Length/Type = Internet Protocol
    - **Network Layer**
      - Version = 4 Hex
      - Header Length = 5 Decimal
      - Precedence = Routine
      - Delay = Normal
      - Throughput = Normal
      - Reliability = Normal
      - Cost = Normal
      - MBZ = 0 Binary
      - Total Length = 124 Decimal
      - Identification = e8a3 Hex
      - Zero = 0 Binary
      - Do not fragment = 0 Binary
      - May fragment = 0 Binary
      - Fragment Offset = 0 Decimal
      - Time To Live = 30 Decimal
      - Protocol = User Datagram Protocol
      - Header Checksum = 8ec9 Hex
      - IP Src Addr = 15.19.03.107 Dot Notation
      - IP Dest Addr = 15.19.03.116 Dot Notation
      - **Transport Layer**
        - Source Port = 2049 Decimal
        - Destination Port = 888 Decimal
        - Length (bytes) = 104 Decimal



Base: Hex

| | +0 | +1 | +2 | +3 |
|---|---|---|---|---|
| | 7 6 5 4 3 2 1 0 | 7 6 5 4 3 2 1 0 | 7 6 5 4 3 2 1 0 | 7 6 5 4 3 2 1 0 |
| Byte 0 | Source Port<br>0x0801 | | Destination Port<br>0x0378 | |
| Byte 4 | Length (bytes)<br>0x0068 | | Checksum<br>0x7D12 | |



Bytes Per Column: 1

Columns Per Row: 4

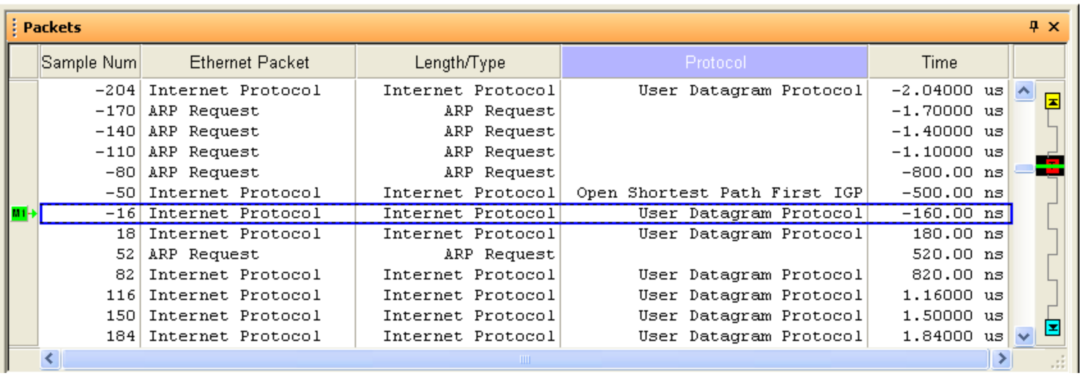
```
0000  5F 75 70 5F   _up_
0004  00 00 00 01   ....
0008  00 00 00 00   ....
000C  00 00 00 00   ....
0010  00 00 00 00   ....
0014  00 00 00 00   ....
0018  00 00         ..
```

## Step 9: Describe the columns displayed in Packet Viewer by default

Finally, use the <DisplayDefaults> element to describe the columns that are displayed in the Packet Viewer by default. For example:

```
<DisplayDefaults>
   <DisplayField Name="Sample Number" Width="16"/>
   <DisplayField Name="Ethernet Packet" Width="25"/>
   <DisplayField Name="Length/Type" Width="20"/>
   <DisplayField Name="Protocol" Width="30"/>
   <DisplayField Name="Time"/>
</DisplayDefaults>
```

These are the default columns displayed when you add the Packet Viewer window:



| Sample Num | Ethernet Packet | Length/Type | Protocol | Time |
|---|---|---|---|---|
| -204 | Internet Protocol | Internet Protocol | User Datagram Protocol | -2.04000 us |
| -170 | ARP Request | ARP Request | | -1.70000 us |
| -140 | ARP Request | ARP Request | | -1.40000 us |
| -110 | ARP Request | ARP Request | | -1.10000 us |
| -80 | ARP Request | ARP Request | | -800.00 ns |
| -50 | Internet Protocol | Internet Protocol | Open Shortest Path First IGP | -500.00 ns |
| -16 | Internet Protocol | Internet Protocol | User Datagram Protocol | -160.00 ns |
| 18 | Internet Protocol | Internet Protocol | User Datagram Protocol | 180.00 ns |
| 52 | ARP Request | ARP Request | | 520.00 ns |
| 82 | Internet Protocol | Internet Protocol | User Datagram Protocol | 820.00 ns |
| 116 | Internet Protocol | Internet Protocol | User Datagram Protocol | 1.16000 us |
| 150 | Internet Protocol | Internet Protocol | User Datagram Protocol | 1.50000 us |
| 184 | Internet Protocol | Internet Protocol | User Datagram Protocol | 1.84000 us |

In the Packet Viewer window, you can insert or delete columns as desired (see "To insert or delete packet decode columns" (in the online help)).

There are four columns that are automatically generated by the Packet Decoder tool:

- "Sample Number" — contains the logic analyzer sample number corresponding to the captured data.
- "Time" — contains the logic analyzer time corresponding to the captured data.
- "(ProtocolFamily name) Packet" — contains the main packet type decodes. For example, if the protocol family name is "Ethernet", the main packet type decodes appear in a column named "Ethernet Packet".

- "Direction" — can contain the name of the Packet Decoder tool or a name you specify in the Packet Decoder tool's properties. This is useful when multiple Packet Decoder tools are used for different directions of a serial link and the tools are named to identify the data direction.

Other columns that can be displayed are the fields described with <Field> elements.

### Getting Started Summary

The previous getting started steps are basic steps for creating a protocol description file. There are additional steps you can take to make your protocol description file more useful. You can:

- Describe protocol errors (see page 44).
- Add color-coding and tool tip descriptions (see page 44).
- You can use ValueFunctions to compute CRC values and look for CRC errors (see page 51).
- You can use TransformFunctions to transform the value of a field (see page 52).

# Adding Decode Information

- "Assigning Meaningful Strings to Values" on page 43
- "Describing Protocol Errors" on page 44
- "Adding Color Descriptions (for Packet Viewer)" on page 44

## Assigning Meaningful Strings to Values

There are two ways to assign meaningful strings to decoded values:

- "Using Enumsets" on page 43
- "Using SymbolDecode (for Lanes tab)" on page 43

### Using Enumsets

When a field can be one of a set of predefined values, use the <Enumset> element to identify those values. For example:

```
<Enumset Name="EthernetV2PacketType">
   <Enum Value="#h0800" Name="Internet Protocol"/>
   <Enum Value="#h0806" Name="ARP Request"/>
   <Enum Value="#h0835" Name="ARP Response"/>
   <Enum Value="#h809b" Name="AppleTalk Datagram"/>
   <Enum Value="#h80d5" Name="SNA"/>
   <Enum Value="#h8137" Name="Novel IPX"/>
   <Enum Value="#h86dd" Name="IPv6"/>
   <Enum Value="#h2007" Name="IPS"/>
   <Enum Value="#h6002" Name="DEC MOP Remote Console"/>
   <Enum Value="#h6004" Name="DEC LAT"/>
</Enumset>
```

Then, use the Enumset attribute in a <Field> element to use the enumeration set:

```
      <Field Name="Length/Type" Length="16"
            Enumset="EthernetV2PacketType"/>
```

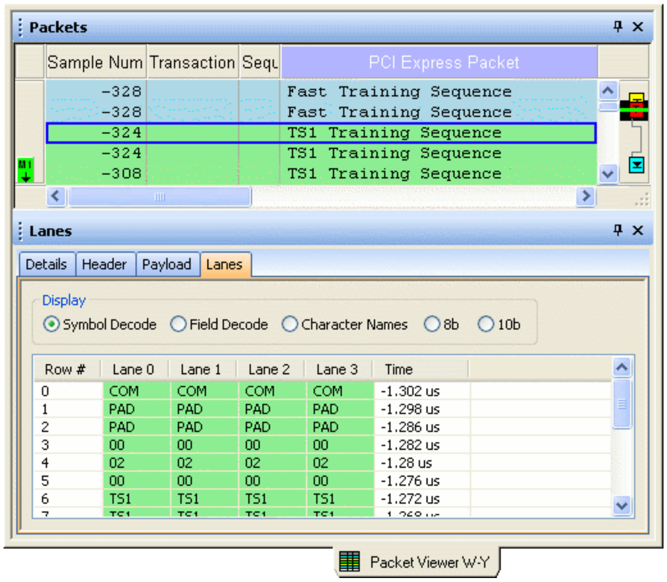
The <Enumset> element can also contain <Range> elements for assigning a string to a range of values and one <Default> element for assigning a string to values not defined by <Enum> or <Range> elements (see the example in "<Default>" on page 75).

### Using SymbolDecode (for Lanes tab)

If your serial protocol uses *lanes* (see page 113), the <SymbolDecode> element is used to assign meaningful strings to decoded values in the Packet Viewer's Lanes tab. For example, the protocol description for PCI Express contains the following <SymbolDecode> assignments.

```
<SymbolDecode>
   <Enumset Name="PCI Express Symbol">
      <Enum Value="#hbc" Name="COM" KDChar="KChar"/>
      <Enum Value="#hfb" Name="STP" KDChar="KChar"/>
      <Enum Value="#h5c" Name="SDP" KDChar="KChar"/>
      <Enum Value="#hfd" Name="END" KDChar="KChar"/>
      <Enum Value="#hfe" Name="EDB" KDChar="KChar"/>
      <Enum Value="#hf7" Name="PAD" KDChar="KChar"/>
      <Enum Value="#h1c" Name="SKP" KDChar="KChar"/>
      <Enum Value="#h3c" Name="FTS" KDChar="KChar"/>
      <Enum Value="#h7c" Name="IDL" KDChar="KChar"/>
      <Enum Value="#h9c" Name="RSV" KDChar="KChar"/>
```

```
            <Enum Value="#hdc" Name="RSV" KDChar="KChar"/>
            <Enum Value="#hfc" Name="RSV" KDChar="KChar"/>
            <Enum Value="#h4a" Name="TS1" KDChar="DChar"/>
            <Enum Value="#h45" Name="TS2" KDChar="DChar"/>
         </Enumset>
      </SymbolDecode>
```



| Sample Num | Transaction | Sequ | PCI Express Packet |
|---|---|---|---|
| -328 | | | Fast Training Sequence |
| -328 | | | Fast Training Sequence |
| -324 | | | TS1 Training Sequence |
| -324 | | | TS1 Training Sequence |
| -308 | | | TS1 Training Sequence |

Display
● Symbol Decode ○ Field Decode ○ Character Names ○ 8b ○ 10b

| Row # | Lane 0 | Lane 1 | Lane 2 | Lane 3 | Time |
|---|---|---|---|---|---|
| 0 | COM | COM | COM | COM | -1.302 us |
| 1 | PAD | PAD | PAD | PAD | -1.298 us |
| 2 | PAD | PAD | PAD | PAD | -1.286 us |
| 3 | 00 | 00 | 00 | 00 | -1.282 us |
| 4 | 02 | 02 | 02 | 02 | -1.28 us |
| 5 | 00 | 00 | 00 | 00 | -1.276 us |
| 6 | TS1 | TS1 | TS1 | TS1 | -1.272 us |

## Describing Protocol Errors

Use the <ProtocolErrors> element to define the protocol errors specified by the protocol that are possible during packet decode. Protocol errors have red highlighting in the Packet Viewer.

For example, to define a protocol error:

```
<ProtocolErrors>
   <ProtocolError Name="Bad Packet"
         Description="The packet ended with the ENB Symbol."/>
</ProtocolErrors>
```

Then, use the ValueError attribute in a <Field> or <Enum> element:

```
<Enumset Name="EndSymbolType">
   <Enum Value="#hfd" Name="END"/>
   <Enum Value="#hfe" Name="EDB" ValueError="Bad Packet"/>
</Enumset>
```

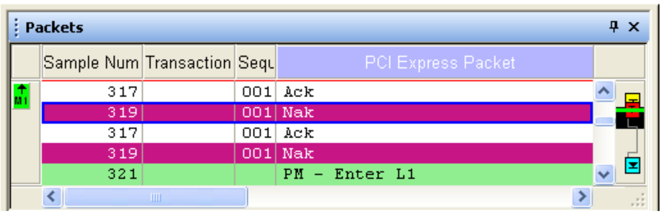
## Adding Color Descriptions (for Packet Viewer)

- “Describing Packet Type Colors" on page 44
- “Describing Cell Highlighting (for Lanes tab)" on page 45
- “Available Colors" on page 46

### Describing Packet Type Colors

Use the <PacketDisplay> element in <PacketType> descriptions. For example:

```
<PacketType Name="Nak">
   <PacketMask Width="12" Value="#h5C1"/>
   <PacketDisplay
         BackgroundColor="MediumVioletRed"
         ForegroundColor="White"
         Description="TLP Sequence Number Negative Acknowledgement.
                      Initiates a Data Link Layer Retry." />
</PacketType>
```



| Sample Num | Transaction | Sequ | PCI Express Packet |
|---|---|---|---|
| 317 | | 001 | Ack |
| 319 | | 001 | Nak |
| 317 | | 001 | Ack |
| 319 | | 001 | Nak |
| 321 | | | PM - Enter L1 |

The Description="(string)" attribute specifies a string that appears in a tool tip when the mouse pointer hovers over the packet type line.

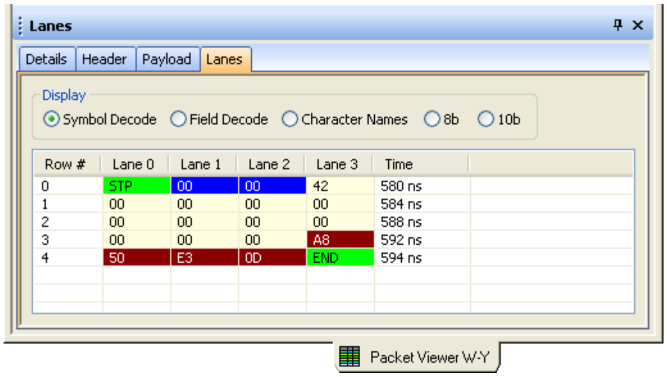
**See Also**

- "Available Colors" on page 46

### Describing Cell Highlighting (for Lanes tab)

Use the <PacketHighlightRules> element to specify cell highlighting in the Packet Viewer's Lanes tab. For example:

```
<PacketHighlightRules>
   <PacketHighlightRule
      PacketSegment="Header"
      ForegroundColor="Black"
      BackgroundColor="LightYellow"
      DisplayName="Header"
   />
   <PacketHighlightRule
      FieldName="Payload"
      FieldType="Payload"
      ForegroundColor="White"
      BackgroundColor="DarkRed"
      DisplayName="Payload"
   />
   <PacketHighlightRule
      FieldName="Sequence Number"
      ForegroundColor="White"
      BackgroundColor="Blue"
      DisplayName="Sequence Number"
   />
   <PacketHighlightRule
      FieldName="LCRC"
      FieldType="CRC"
      ForegroundColor="Black"
      BackgroundColor="Gray"
      DisplayName="LCRC"
   />
   <PacketHighlightRule
      FieldName="TLP Digest (ECRC)"
      FieldType="CRC"
      ForegroundColor="Black"
      BackgroundColor="LightGrey"
      DisplayName="ECRC"
   />
   <PacketHighlightRule
      FieldName="16b CRC"
      FieldType="CRC"
      ForegroundColor="Black"
      BackgroundColor="LightGrey"
      DisplayName="16b CRC"
   />
   <PacketHighlightRule
      LaneKDChar="KChar"
      ForegroundColor="Black"
      BackgroundColor="Green"
   />
</PacketHighlightRules>
```



Display
● Symbol Decode ○ Field Decode ○ Character Names ○ 8b ○ 10b

| Row # | Lane 0 | Lane 1 | Lane 2 | Lane 3 | Time |
|---|---|---|---|---|---|
| 0 | STP | 00 | 00 | 42 | 580 ns |
| 1 | 00 | 00 | 00 | 00 | 584 ns |
| 2 | 00 | 00 | 00 | 00 | 588 ns |
| 3 | 00 | 00 | 00 | A8 | 592 ns |
| 4 | 50 | E3 | 0D | END | 594 ns |

**See Also**

- "Available Colors" on page 46
- "<PacketHighlightRules>" on page 93
- "<PacketHighlightRule>" on page 91

## Available Colors

You can choose from the following available colors:

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| ⬛ | Black | 🟧 | DarkOrange | 🟩 | PaleGreen | 🟦 | SlateGray |
| ⬛ | DimGray | 🟫 | BurlyWood | 🟩 | LightGreen | 🟦 | LightSteelBlue |
| ⬛ | Gray | ⬜ | AntiqueWhite | 🟩 | ForestGreen | 🟦 | CornflowerBlue |
| ⬜ | DarkGray | 🟫 | Tan | 🟩 | LimeGreen | 🟦 | RoyalBlue |
| ⬜ | Silver | ⬜ | NavajoWhite | 🟩 | DarkGreen | ⬜ | GhostWhite |
| ⬜ | LightGrey | ⬜ | BlanchedAlmond | 🟩 | Green, Lime | ⬜ | Lavender |
| ⬜ | Gainsboro | ⬜ | PapayaWhip | 🟩 | SeaGreen | 🟦 | MidnightBlue |
| ⬜ | WhiteSmoke | ⬜ | Moccasin | 🟩 | MediumSeaGreen | 🟦 | Navy |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| ⬜ | White | 🟧 | Orange | 🟩 | SpringGreen | 🟦 | DarkBlue |
| ⬜ | Snow | 🟨 | Wheat | ⬜ | MintCream | 🟦 | MediumBlue |
| 🟫 | RosyBrown | ⬜ | OldLace | 🟩 | MediumSpringGreen | 🟦 | Blue |
| 🟥 | LightCoral | ⬜ | FloralWhite | 🟩 | MediumAquamarine | 🟦 | SlateBlue |
| 🟥 | IndianRed | 🟫 | DarkGoldenrod | 🟩 | Aquamarine | 🟦 | DarkSlateBlue |
| 🟥 | Brown | 🟨 | Goldenrod | 🟦 | Turquoise | 🟪 | MediumSlateBlue |
| 🟥 | FireBrick | ⬜ | CornSilk | 🟦 | LightSeaGreen | 🟪 | MediumPurple |
| 🟫 | Maroon | 🟨 | Gold | 🟦 | MediumTurquoise | 🟪 | BlueViolet |
| 🟥 | DarkRed | 🟨 | LemonChiffon | ⬜ | Azure | 🟦 | Indigo |
| 🟥 | Red | 🟨 | Khaki | ⬜ | LightCyan | 🟪 | DarkOrchid |
| ⬜ | MistyRose | 🟨 | PaleGoldenrod | 🟦 | PaleTurquoise | 🟪 | DarkViolet |
| 🟧 | Salmon | 🟨 | DarkKhaki | ⬛ | DarkSlateGray | 🟪 | MediumOrchid |
| 🟥 | Tomato | ⬜ | Ivory | 🟦 | Teal | 🟪 | Thistle |
| 🟧 | DarkSalmon | ⬜ | Beige | 🟦 | DarkCyan | 🟪 | Plum |
| 🟧 | Coral | ⬜ | LightYellow | 🟦 | Aqua, Cyan | 🟪 | Violet |
| 🟧 | OrangeRed | ⬜ | LightGoldenrodYellow | 🟦 | DarkTurquoise | 🟪 | Purple |
| 🟧 | LightSalmon | 🟫 | Olive | 🟦 | CadetBlue | 🟪 | DarkMagenta |
| 🟫 | Sienna | 🟨 | Yellow | 🟦 | PowderBlue | 🟪 | Fuchsia, Magenta |
| ⬜ | Seashell | 🟩 | OliveDrab | 🟦 | LightBlue | 🟪 | Orchid |
| 🟧 | Chocolate | 🟩 | YellowGreen | 🟦 | DeepSkyBlue | 🟥 | MediumVioletRed |
| 🟫 | SaddleBrown | 🟩 | DarkOliveGreen | 🟦 | SkyBlue | 🟥 | DeepPink |
| 🟧 | SandyBrown | 🟩 | GreenYellow | 🟦 | LightSkyBlue | 🟥 | HotPink |
| 🟧 | PeachPuff | 🟩 | Chartreuse | 🟦 | SteelBlue | ⬜ | LavenderBlush |
| 🟧 | Peru | 🟩 | LawnGreen | ⬜ | AliceBlue | 🟥 | PaleVioletRed |
| ⬜ | Linen | ⬜ | Honeydew | 🟦 | DodgerBlue | 🟥 | Crimson |
| ⬜ | Bisque | 🟩 | DarkSeaGreen | 🟦 | LightSlateGray | 🟥 | Pink |

## How to ...

- “To decode conditionally based on packet bits" on page 48
- “To determine serial data start of packet by using look around" on page 48
- “To display and use full values for partial bit fields" on page 49
- “To decode fields with printf-style format strings" on page 49
- “To add information to a packet" on page 50

### To decode conditionally based on packet bits

Make <Field> element decoding conditional by using the Select="(formula)" attribute. For example:

```
      <Field Name="Internet Cntrl Msg Protocol" Type="Protocol"
            Select="'Protocol'==#h01"/>
      <Field Name="Transmission Control Protocol" Type="Protocol"
            Select="'Protocol'==#h06"/>
      <Field Name="User Datagram Protocol" Type="Protocol"
            Select="'Protocol'==#h11"/>
      <Field Name="Open Shortest Path First IGP" Type="Protocol"
            Select="'Protocol'==#h59"/>
   </Header>
</Protocol>
```

In this example, the fields are only decoded if the previously defined Protocol field has the specified value. You can also look ahead to decode conditionally based on fields later in the packet (see “To look ahead" on page 58).

### To determine serial data start of packet by using look around

Use the look around syntax in formulas. By appending a {} to a bus/signal name in a formula, data can be retrieved from previous or future samples.

For example, if the bits "11110000" on a single serial channel identify the start-of-packet, you could use:

```
<Bus Name="Serial Bus" Style="Parallel">
   <Label Name="My TXD" Width="1" Type="Data"
         Sop="'My TXD'{-7}==#b1 .land.
              'My TXD'{-6}==#b1 .land.
              'My TXD'{-5}==#b1 .land.
              'My TXD'{-4}==#b1 .land.
              'My TXD'{-3}==#b0 .land.
              'My TXD'{-2}==#b0 .land.
              'My TXD'{-1}==#b0 .land.
              'My TXD'{0}==#b0"
   <Protocol Name="Serial Bus Protocol" Type="Packet"/>
</Bus>
```

**NOTE** **The Sop attribute formula string must appear on one line. The formatting in the example above is for readability.**

### To display and use full values for partial bit fields

The <Field> element's DecodeRule, EncodeRule, and DisplayLength attributes let you display and use full values when not all bits of a field are transmitted in a packet.

For example, in PCI Express, there is an Address[31:2] field in a packet. You probably want to view the value as a full 32-bit value. Also, when searching, triggering, or filtering on that field, you would like to enter 32-bit values and have the software "normalize" that 32-bit value back to what it should be in the packet. In this case, you can use:

```
<Field Name="Address" Type="Address" Length="30"
      DecodeRule="Address .lshift. 2"
      EncodeRule="Address .rshift. 2"
      DisplayLength="32"/>
```

The DecodeRule attribute specifies how the value of the field should be transformed prior to display. In the preceding example, it is to left shift the value of the field by 2 bits.

The EncodeRule attribute specifies how the entered value when triggering, searching, or filtering on the field should be transformed before performing the operation. In the preceding example, it is to right shift the value of the field by 2 bits.

Finally, the DisplayLength attribute specifies the length of the value to display (after the transform has occurred). In the preceding example, it is 32 bits.

### To decode fields with printf-style format strings

The <Field> element's DecodeString attribute lets you format decoded fields using C language printf-style format strings. This gives you great flexibility in formatting decoded data.

For example, this decode string displays length data separated by colons:

```
<Field Name="Length" Length="8"
      DecodeString="'%d:%d:%d:%d', 'Length'[10:9], 'Length'[8:7],
      'Length'[6:4], 'Length'[3:0]"
      Select="'Packet Type'==#h1"/>
```

**NOTE** **The DecodeString attribute string must appear on one line. The formatting in the example above is for readability.**

Only integers are to be used in the printf specification; therefore, these data types are supported:
- c (single-byte character).
- C (wide character).
- d, i (signed decimal integer).
- o (unsigned octal integer).
- u (unsigned decimal integer).
- x (unsigned hexadecimal integer, using "abcdef").
- X (unsigned hexadecimal integer, using "ABCDEF").

There is no support for floating point values (e,E,f,g,G), pointer values (n,p), or strings (s,S).

The full C-Printf syntax is allowed, except for the above data types.

New lines are not displayed.

### To add information to a packet

You can use the <Label> element's Meta attribute to add additional information to a packet. This additional information, for example, can be used to:

- Qualify the selection of packet types.
- Provide additional decode information.
- Selectively decode fields in a different way.

Meta data is specified as a concatenation of labels or constants.

The underlying structure of a meta data specification is a buffer. As each label or constant is read in, the buffer is filled in with the corresponding data. The labels or constants are delimited by commas.

Each label or constant token must be appended with a backslash followed by the number of bits that should be added to the buffer.

As an example, here is a meta specification that includes 2 bits of a 4-bit status label:

```
<Label Name="Status" Width="4" />
<Label Name="Data" Sop="Status==1" Meta="'Status'[4:3]\2" />
```

Here is another example that includes the 4 bits of status, appended with 2 bits of a constant (the two bits to be added to the buffer is 3 or 11b):

```
<Label Name="Status" Width="4" />
<Label Name="Data" Sop="Status==1" Meta="Status\4,3\2" />
```

Note that, in these examples, the meta data is currently added only on the SOP state. You could add {-n} to the label name to get a value n states before the SOP state, or you could add {n} to get values n states after the SOP state.

## Using Advanced Features

- "Using ValueFunctions" on page 51
- "Using TransformFunctions" on page 52
- "When the Framing Options are Not Sufficient" on page 53

### Using ValueFunctions

The <Field> element's ValueFunction attribute lets you use an external program to calculate the expected value of a field. ValueFunction attributes are normally used for CRC computation. For example:

```
         <Trailer>
            <Field Name="Next Control Word Data" Length="12"/>
            <Field Name="DIP-4" Length="4"
                  ValueFunction="agProtocols:SPI42DIP4"
                  ValueInput="DIP-4 Contents"
                  ValueError="Bad DIP-4"/>
         </Trailer>
      </FieldContainer>
   </Protocol>
```

The ValueFunction attribute's value takes the form "LibraryName:FunctionName" where the library is a DLL with unmangled names. For example, here is the forward declaration of a value function:

```
extern "C" __declspec( dllexport )
   unsigned __int32 FunctionName( unsigned __int8* pData_p,
                                  unsigned __int32 nByteLength_p );
```

The '"C" __declspec( dllexport )' part of the declaration is for unmangled names.

The Packet Decoder tool passes in a pointer to the data identified by the ValueInput attribute and its byte length.

The function returns a 32-bit unsigned integer that is the calculated field value. A protocol error occurs if the actual field value is different than the returned value.

Any C compiler can be used to generate the DLL as long as the function names are unmangled.

DLL files (and any additional DLL files referenced by the ValueFunction) should be located in the Protocols folder in the *Keysight Logic Analyzer* application's install directory. For example, the default location is: C:\Program Files\Keysight Technologies\Logic Analyzer\Protocols.

**Example** Here is the ValueFunction used for SPI 4.2 to perform a DIP-4 Parity check:

```
extern "C" __declspec( dllexport )
    unsigned __int32 SPI42DIP4( unsigned __int8* pData_p,
                                unsigned __int32 nLength_p )
{
  // need to mask the DIP-4 value to all 1's.  This value is the
  // last 4 bits of the packet
  pData_p[ nLength_p - 1 ] |= 0x0F;

  // This routine works better if we can address the bytes in pairs.
  unsigned __int16* pData
      = reinterpret_cast< unsigned __int16* >( pData_p );
  unsigned __int32  nLines = nLength_p / 2;

  unsigned __int32 nDIP4 = 0;
  for ( unsigned __int32 nLine = 0; nLine < nLines; ++nLine )
  {
    unsigned __int32 nUpperBits
        = pData[nLine] << ( nLine % 16 );
    unsigned __int32 nLowerBits
        = pData[nLine] >> ( 16 - ( nLine % 16 ) );
    unsigned __int32 nShiftedData
        = ( nUpperBits | nLowerBits ) & 0x0ffff;

    nDIP4 ^= nShiftedData;
  }

  unsigned __int32 nUpperDIP4 = ( nDIP4 & 0xff00 ) >> 8;
  unsigned __int32 nLowerDIP4 = ( nDIP4 & 0x00ff );

  nDIP4 = (nUpperDIP4 ^ nLowerDIP4) & 0xFF;

  nUpperDIP4 = ( nDIP4 & 0xF0 ) >> 4;
  nLowerDIP4 = ( nDIP4 & 0x0F );

  nDIP4 = ( nUpperDIP4 ^ nLowerDIP4 ) & 0x0F;

  return nDIP4;
}
```

### Using TransformFunctions

The <Field> element's TransformFunction attribute lets you use an external program to transform the value of a field. For example, this would be required to modify the order of the bits/bytes of a field:

```
<Protocol Name="I/O Cycle" ProtocolLayer="Physical">
   <Field Name="Address" Length="16"/>
   <Field Name="Payload" Length="8" Type="Payload"
         TransformFunction="agProtocols:LPCSwapNibbles"/>
</Protocol>
```

The TransformFunction attribute's value takes the form "LibraryName:FunctionName" where the library is a DLL with unmangled names. For example, here is the forward declaration of a transform function:

```
extern "C" __declspec( dllexport )
   unsigned __int32 FunctionName( unsigned __int8* pData_p,
                                  unsigned __int32 nByteLength_p );
```

The '"C" __declspec( dllexport )' part of the declaration is for unmangled names.

The Packet Decoder tool passes in a pointer to the field data and its byte length.

The function returns a 32-bit unsigned integer that is not used.

Any C compiler can be used to generate the DLL as long as the function names are unmangled.

DLL files (and any additional DLL files referenced by the TransformFunction) should be located in the Protocols folder in the *Keysight Logic Analyzer* application's install directory. For example, the default location is: C:\Program Files\Keysight Technologies\Logic Analyzer\Protocols.

**Example** Here is the TransformFunction used for LPC to perform nibble swapping:

```
extern "C" __declspec( dllexport )
    unsigned __int32 LPCSwapNibbles( unsigned __int8* pData_p,
                                     unsigned __int32 nLength_p )
{
  for ( unsigned __int32 nByte = 0; nByte < nLength_p; ++nByte )
  {
    //
    // for each byte, swap nibbles
    //
    unsigned __int8 nHighNibble = ( pData_p[nByte] >> 4 ) & 0x0F;

    pData_p[nByte] = ( pData_p[nByte] << 4 ) & 0x0F0 | nHighNibble;
  }

  return 0;
}
```

## When the Framing Options are Not Sufficient

There are some cases where the built-in frame detection options of the protocol description file are not sufficient for detecting the start- and end-of-packets. In some of these cases, you can use other logic analysis system tools to massage the data before it gets to the Packet Decoder tool.

- “Using the Serial To Parallel Tool" on page 53
- “Using the Signal Extractor Tool" on page 53

### Using the Serial To Parallel Tool

When serial data is captured without a clock signal in the timing (asynchronous) sampling mode, you can use the Serial To Parallel tool to extract a clock signal from the data. Then, you can tell where bit values start and end, and you can pass that information on to the Packet Decoder tool.

For more information, see "Using the Serial To Parallel Tool" (in the online help).

### Using the Signal Extractor Tool

For speed reasons, data is some times captured in demultiplexed form on multiple buses/signals. The Packet Decoder tool cannot decode data from two buses/signals. However, you can use the Signal Extractor tool to remultiplex data before it is processed by the Packet Decoder tool.

For more information, see "Using the Signal Extractor Tool" (in the online help).

# 3 Using Formulas

## Using Formulas in Bus/Signal Label Descriptions

Formulas contain a simple meta-language used to make the execution of decoding and framing more dynamic.

- "To determine the start-/end-of-packets" on page 56
- "To determine valid data" on page 56
- "To look around" on page 56
- "To identify rising/falling/toggling signals" on page 56

### To determine the start-/end-of-packets

Formulas let you frame data based upon the current values of buses/signals.

<Label> element formulas can operate on other bus/signal values that have been defined within the <Bus> element.

```
<Bus Name="TestBus">
   <Label Name="Control" Width="1" Type="Frame" />
   <Label Name="Main" Width="8" Type="Data" Sop="Control==#h1" />
   <BusProtocol ... />
</Bus>
```

The Sop formula for the Type="Data" label is shown above. The formula uses the "Control" bus/signal to determine if the current state is a start-of-packet.

### To determine valid data

Formulas let you look at the current values of buses/signals to determine when data is valid.

<Label> element formulas can operate on other bus/signal values to determine when data is valid.

```
<Bus Name="TestBus">
   <Label Name="Control" Width="1" Type="Frame" />
   <Label Name="Extra" Width="4" Type="Valid" />
   <Label Name="Main" Width="8" Type="Data" Sop="Control==#h1"
         Valid="'Extra'[0]==#b0" />
   <BusProtocol ... />
</Bus>
```

The Valid formula for the Type="Data" label is shown above. The formula uses one of the "Extra" signals to determine if the data is valid.

### To look around

Additionally, minimal functionality has been added to provide some look around capability. By appending a {} to a bus/signal name in a formula, data can be retrieved from previous or future samples. For example:

```
Sop="Control{-1}==#h1 .land. Control{0}==0"
```

If more complicated look around capability is required to determine the start-/end-of-packet, an inverse assembler can be used to generate the required SOP/EOP signals (see "To develop your own tools" (in the online help)).

### To identify rising/falling/toggling signals

Some predefined terms can be also used to simplify specification of common terms in framing.

Rising, Falling, and Toggling are terms that can be appended to bus/signal names in a formula as well. For example:

```
Sop="Control Rising"
Sop="Control Falling"
Sop="Control Toggling" - This means a start-of-packet occurs at each
                         transition of "Control".
```

A wide variety of additional operators are available for use in formulas, for more information, see Chapter 7, “Formula Reference,” starting on page 109.

## Using Formulas in Field Descriptions

Formulas are a simple meta-language that are used to make the execution of decoding and framing more dynamic. The use of a formula allows the decoding logic to be dependent upon the current values of packet data.

- “To operate on other field values" on page 58
- “To look ahead" on page 58
- “To get the length of variable-length packets" on page 58

### To operate on other field values

Field element formulas let you specify other field names as inputs to the formula. For example:

```
<Protocol Name="Control Packet" ProtocolLayer="Data">
   <Field Name="BufferLength" Length="10"/>
   <Field Name="Buffer" Length="BufferLength*8"/>
</Protocol>
```

The Field formula for the Buffer field is shown above. The formula uses the previously specified BufferLength field to extract the value and use it to compute the length of the Buffer field.

### To look ahead

Usually, the field needed to calculate the formula is before the field that typically requires it. However, sometimes that is not the case. The use of a lookahead field is required to extract the value of the future field.

```
<Protocol Name="Transmit Packet" ProtocolLayer="Data">
   <Field Name="PacketTypeLookahead" Length="1" BitOffset="10"
         Type="Lookahead"/>
   <Field Name="ControlPacket" Type="Protocol"
         Select="PacketTypeLookahead==#h0"/>
   <Field Name="PayloadPacket" Type="Protocol"
         Select="PacketTypeLookahead==#h1"/>
   <Field Name="PacketType" Length="1"/>
</Protocol>
```

In this example, a packet that has a packet type field 10 bits into the packet. However, to decode the first 10 bits, the packet type field must be known to properly decode the packet. In order to accomplish this, a lookahead field is used to prematurely offset into and extract the field data. This field is an internal, hidden field that will not be shown in any of the Packet Viewer windows. It can, however, be used for formulas. Based on the value of the lookahead field, the packet type can be determined and the decoding works as expected.

### To get the length of variable-length packets

Sometimes, it is useful to know the length of the packet when computing formulas. This is particularly true for length formulas in payload fields. Many times, the length of the payload is variable, with no length field to specify the actual length of the payload.

To help address this situation, a special "constant" is available which returns the bit length of the entire *framed* packet (that is, from the start-of-packet to the end-of-packet, if defined, or to the next start-of packet if no end-of-packet is defined). This constant is '#PACKET_LENGTH'. For example:

```
<Field Name="Payload" Length="'#PACKET_LENGTH' - 32" Type="Payload"/>
```

In the previous example, the length of the payload field is the total length of the packet minus 32 bits for the header and trailer.

# 4 Solving Problems

## Protocol Description Errors when Application Starts

### General Errors

General errors before any real parsing is done (for example, checking for well-formedness).

#### Cannot find decryption function for: 'filename'

You may need to reinstall the *Keysight Logic Analyzer* application as it is responsible for depositing the decryption library.

#### Decryption of 'filename' failed: {parsing errors}

This error occurs when opening files that are not .aex format files.

#### Malformed XML in 'filename'

Occurs when XML markup in the file is not well formed.

#### XML Parser Load of 'filename' failed: {parsing errors}

### Parsing Errors

All parsing errors have a standard "location" appended. This "location" is of the form:

```
Error: {strMessage}
Location: <ProtocolFamilyName><Element Name="">
```

#### <PacketType> has no specified protocol.

This message occurs when there is no Protocol attribute. The Protocol attribute is optional for the <PacketTypeGroup>, <PacketTypes>, and <PacketType> elements; the error occurs when the attribute is not supplied with any of these elements.

#### Cannot find 'Default' packet type '{value}.

For the Default attribute in the <PacketTypes> element.

#### Cannot find 'Enumset' reference: {name}.

For the Enumset attribute in the <Field> element.

#### Cannot find 'Name' protocol reference: {Name}.

For the Name attribute in the <BusProtocol> element.

#### Cannot find 'ValueError' reference: {name}.

For the ValueError element in the <Field> element.

#### Cannot find default packet type with name: '{Default}'.

For the Default attribute in the <PacketTypes> element.

#### Cannot find referenced protocol '{Protocol}' name.

For the Protocol attribute in the <PacketType>, <PacketTypes>, and <PacketTypeGroup> elements.

#### Duplicate <{Element}> elements found with name '{name}'.

For the <Enumset>, <PacketTypes>, and <Protocol> elements.

#### Error in loading library GetLastError() = {Error}, Path={Path}

For the ValueFunction attribute in the <Field> element.

**Missing required attribute '{Attribute}' in element <{Element}>.**

Note that in the <Field> element:

- The Length attribute is not required when Type="Protocol" or Type="ProtocolField".
- The ValueInput attribute is only required if the ValueFunction attribute is specified.

**More than one <Default> element present in {EnumSet:Name}. Additional <Default> elements will be ignored.**

**No <BusProtocol> elements found under <Bus>.**

The <Bus> element must contain a <BusProtocol> element.

**No <Label> elements found under <Bus>.**

The <Bus> element must contain at least one <Label> element.

**No segments defined for <SegmentedField>.**

To be useful, the <SegmentedField> element should contain <Segment> elements.

**Only one <{Element}> element is permitted under the <ProtocolFamily> element.**

The <ProtocolFamily> element can contain one each of the following elements: <DisplayDefaults>, <PacketHighlightsRule>, <ProtocolErrors>, and <SymbolDecode>.

**The color specified '{value}' for attribute '{Attribute}' is invalid.**

Can occur with the <PacketHighlightRule> and <PacketTypeDisplay> elements' BackgroundColor and ForegroundColor attributes (see "Available Colors" on page 46).

**The following error(s) occurred while parsing '{Attribute}' attribute formula:**

Can occur with these attributes that allow a formula value:

- The AbsoluteBitOffset attribute in the <Field> element.
- The Length attribute in the <Field>, <FieldGroup>, and <RepetitiveField> elements.
- The Select attribute in the <BusProtocol> and <Field> elements.
- The Value attribute in the <Field> element.

These messages have additional error content generated by the formula parser. Formula parser errors are usually some kind of syntax error in the formula description. Formula parser errors are usually self-explanatory - typically either a token is missing or unexpected.

**Unable to find function '{ValueFunction}' in library '{Library}' for <Field> '{name}'.**

For the ValueFunction attribute in the <Field> element.

Make sure the library referenced has the specified function and the function is being properly exported (with the dllexport, see "Using ValueFunctions" on page 51).

**Unable to find library with name '{name}' for field '{name}'.**

For the ValueFunction attribute in the <Field> element.

Make sure the .dll with the specified name is located in the Protocols folder in the *Keysight Logic Analyzer* application's install directory (for example, the default location is: C:\Program Files\Keysight Technologies\Logic Analyzer\Protocols).

It is also possible to get this error if you have linked with another dll that is missing from the Protocols folder.

**Unable to parse formula in 'Value' attribute.**

For the <PacketTypeMask> element.

**Unknown value '{value}' for attribute '{Attribute}' in element {Element}.**

This error occurs when an attribute value is not one of the predefined set of possible values.

**Unsupported version. 'Version' attribute in <ProtocolFamily> element must be 1.1.**

**Value of 'Width' attribute must less than or equal to 32.**

For the <PacketTypeMask> element.

## Decode Errors

Decode errors appear only in Packet Viewer window tool tips.

**Cannot decode '{Attribute}' attribute formula for <{Element} Name='{name}>.**

This error occurs during decode, typically when a formula is not able to be parsed because a field referenced in a formula is not present in the decoded packet.

**Zero Length field for <Protocol>: {name}, <Field>: {name}.**

This error occurs during decode.

## Pre-Defined Protocol Errors that Appear in Packet Viewer

### Unknown Packet Type

This means none of the defined packet type values match the value of the captured packet. You can add the appropriate packet type description to make this error go away (see "Step 7: Describe the packet types" on page 33).

### Unexpected End Of Packet

There are several possible causes:

- When the next start-of-packet occurs before the full number of bits in the packet have been decoded.
- When not enough data is being framed into a packet. This usually means the framing information is not set up correctly under the <Bus> element.

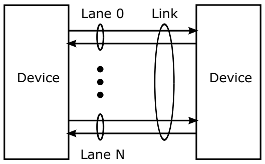
  A good way to look at all of the data that has been framed into a packet is to use the "Packet Bytes" base on the "(ProtocolFamily name) Packet" column in the Packet Viewer window. This shows all bytes of the packet. If not all of the data in the packet is present, a problem with the SOP/EOP formulas is likely.
- When a variable-length field's length is being incorrectly calculated.

  Make sure that the Length attribute in each field correctly specifies the number of bits.

# 5 Multi-Lane Serial Link Concepts

A single physical channel (or lane) is a full-duplex serial connection, in other words, a single transmit/receive pair.

To achieve greater bandwidths, a serial protocol can use multiple lanes in the connection between two devices. The multi-lane connection is called a link.



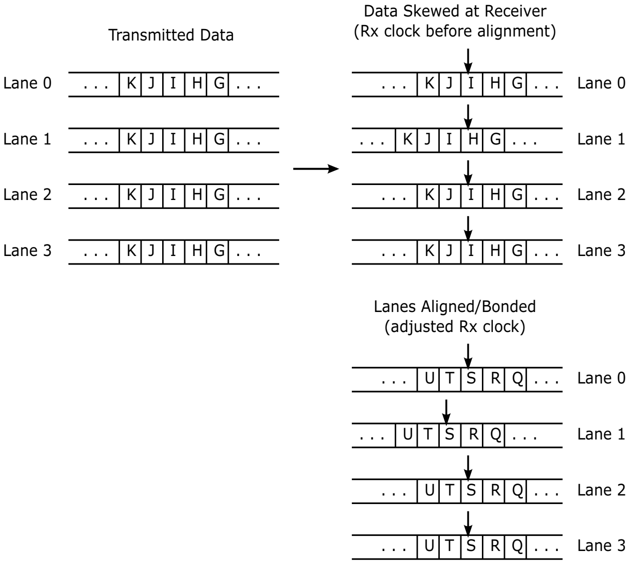
The number of serial channels in a link are sometimes described with a number and an "x". For example, InfiniBand can have 1X, 4X, and 12X links; PCI Express can have x1, x2, x4, x8, and x16 links.

## Lane Initialization

Before data can be transmitted and received over a link between devices, communication within the individual lanes must first be established. This is known as lane initialization.

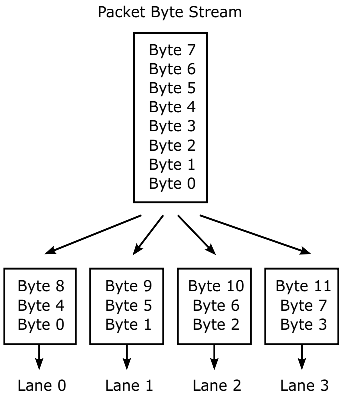
## Aligning (Bonding) Lanes

After the individual lanes in a link have been initialized, the lanes must be aligned (or bonded).



## Striping Packet Data Across Lanes

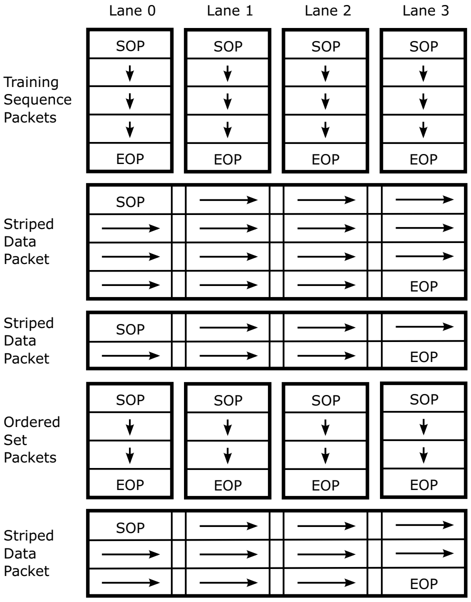
Once lanes are bonded, packet data can be striped across the lanes to achieve greater data transfer rates.



The previous example shows data striped in bytes, but it can also be striped in byte pairs or other data lengths.

## Training Sequence and Ordered Set Packets

Sometimes, training sequence packets and ordered set packets are inserted between striped data packets. These packets are sent on all the lanes; they are not striped across the lanes.



When decoding multi-lane links, the protocol description file's OrderedSetSop attribute in the <Label> element tells the decoder when packets switch from being striped to being sent on all lanes. Packets that are not striped have their own lane decode protocol descriptions.

## Probing a Multi-Lane Serial Link with a Logic Analyzer

Special analysis probe hardware can be used to capture data on a link. An analysis probe recovers clock signals, deserializes data for each lane, decelerates the clock, and provides parallel data for each lane to the logic analyzer, along with other control signals. The logic analyzer probe pods capture the parallel data.

Of course, there are other ways to probe multi-lane serial links, depending on the link's speed and probe points in the device under test.

## 8B and 10B Buses From the Probe

Many serial protocols use 8B/10B encoding. An analysis probe deserializes the 10B data and provides it to the logic analyzer in parallel form. An analysis probe can also decode the 10B data to 8B data. The 8B data and the 10B data are typically provided to the logic analyzer using the same signals, so there are also control signals that tell the logic analyzer when the data is in 10B or 8B form.

## Logical Lanes vs. Physical Lanes

Physical lanes are the number of physical channels in a link.

A link, regardless of the number of physical channels, can be contain a number of independent data streams. These are logical lanes, and they are not related to the number of physical lanes.

Customizing Protocol Descriptions for Packet Viewer

Online Help

# 6 XML Element Reference

## <Bus>

This element describes a bus or a grouping of buses/signals (as defined by a logic analyzer module or tool) that are required for this protocol.

**Required** Yes

**Attributes**

| Name | Required | Value | Comment |
|---|---|---|---|
| Name | yes | string | Name that will be shown in the Packet Decoder's list of buses that can be decoded with this protocol. |
| GenerateLaneData | no | "T" | Enables the generation of the "(ProtocolFamily name) Lane Data" column in the Listing window. See "Step 6: Describe the bus to be decoded" on page 27. |
| | | "F" (default) | Disables the generation of the "(ProtocolFamily name) Lane Data" column in the Listing window. |
| LogicalLanes | no | integer (default: "1") | The number of lanes actually being analyzed. For example, in the N4220B PCI Express packet analysis probe, even though 4 lanes are being probed, there may only be 1 or 2 lanes being analyzed. |
| MaxSearchStates | no | integer (default: "4096") | This specifies the number of samples in which to look for EOP from the SOP. The default is 4096 samples. |
| PhysicalLanes | no | integer (default: "1") | The number of lanes being probed. For example, in the N4220B PCI Express packet analysis probe, x1, x2, and x4 data is decelerated to span 4 buses/signals. |
| ProtocolBits | no | integer (default: "0") | The number of bits needed in a packet before the protocol type is determined. See <Protocol> Element. |
| SOPEOPOnSameSymbol | no | "T" | This must be set if a protocol requires EOP to be checked on the same symbol as SOP. |
| | | "F" (default) | SOP and EOP are not checked on the same symbol in the same state. |

**Contains**

- "<Label>" on page 87 (required)
- "<BusProtocol>" on page 73 (required)

**Contained By**

- "<ProtocolFamily>" on page 102

**Example** This specifies a demux of 4-1:

```
<Bus Name="My Rx Bus" LogicalLanes="4" PhysicalLanes="1"/>
```

The attempt to find a packet type match will not occur until 32 bits of data has been framed for the current packet:

```
<Bus Name="My Rx Bus" LogicalLanes="1" PhysicalLanes="1"
      ProtocolBits="32"/>
```

## <BusProtocol>

This element specifies the initial protocol to use when decoding the bus. A formula can be used to select between several different protocols. Also, multiple protocols may exist to perform packet decoding, *symbol decoding* (see page 113), and lane or *ordered set* (see page 113) decoding. The type of decoding to be done is specified in the Type attribute.

**Required** Yes

**Attributes**

| Name | Required | Value | Comment |
|---|---|---|---|
| Name | yes | string | Name of the root protocol or symbol decode to use in decoding. |
| Type | yes | "Packet" | Specifies that normal, single-channel (or lane) packet decoding will take place for the bus. The decoding is described by a <Protocol> element (see page 99) with a matching Name attribute. |
| | | "Lane" | Specifies that ordered set decoding, used with multi-lane protocols, will take place for the bus. The decoding is described by a <Protocol> element (see page 99) with a matching Name attribute. See also Chapter 5, "Multi-Lane Serial Link Concepts," starting on page 67. |
| | | "Symbol" | Specifies that symbol decoding, where data values are translated to symbolic names, will take place for the bus. The decoding is described by a <SymbolDecode> element (see page 107) that contains an <Enumset> element (see page 79) with a matching Name attribute. |
| FixedLength | no | integer (default: "0") | Length of the protocol in bits. A zero means the protocol is not fixed length. |
| InterruptiblePacket | no | formula (see page 55) | The formula specifies packets that can be interrupted by others. Interruptible packets are continued after the interrupting packet. |
| InterruptingPacket | no | formula (see page 55) | The formulas specifies packets that can interrupt others. |
| Select | no | formula (see page 55) | Used to determine if the protocol should be used for the current packet. The contents of the formula can include references to the data from the current packet and must include a range or index operator [ ]. |

**Contains** None

**Contained By**

- "<Bus>" on page 72

**Example**

```
<Bus>
   <Label ...>
   <Label ...>
   <BusProtocol Name="Rx Packet" Type="Packet" Select="data[0]==1"/>
   <BusProtocol Name="Tx Packet" Type="Packet" Select="data[0]==0"/>
</Bus>
```

**NOTE**

The use of a range or index operator [ ] in the Select formula is required because extractions are occurring from the packet data itself. No field names can be used to extract the data, because the decode operation has not taken place yet. The name used to specify the data is arbitrary and any name can be used.

# <Default>

This element assigns a meaningful string to a value that is not described by <Enum> or <Range> elements in the <Enumset>.

**Required** No

**Attributes**

| Name | Required | Value | Comment |
|---|---|---|---|
| Name | yes | string | The name assigned to a value that is not described by <Enum> or <Range> elements. |
| Description | no | string | A description of the default. |
| ValueError | no | string | If present, the ValueError attribute specifies that this default value is an error and will appear in Red in the Packet Viewer window. A matching string must appear in the <ProtocolErrors> element (see page 101). See "Describing Protocol Errors" on page 44. |
| KDChar | no | "Kchar", "Dchar", or "DontCare" | Specifies if the value is a K character or D character. |

**Contains** None

**Contained By**

- "<Enumset>" on page 79
- "<Field>" on page 80 (Default can be added inline to field definitions as a shortcut to creating an enumset external to the protocol. See the example in "<Enum>" on page 78.)

**Example**

```
<Enumset Name="FirstValueSet">
   <Enum Name="OK" Value="0"/>
   <Enum Name="Error" Value="1"/>
   <Enum Name="Busy" Value="2"/>
   <Enum Name="Retry" Value="3"/>
   <Range Name="Bad Value" LowValue="8" HighValue="15"
         ValueError="Bad Value Description"/>
   <Default Name="Good Default"/>
</Enumset>

<Enumset Name="SecondValueSet">
   <Enum Name="OK" Value="0"/>
   <Enum Name="Error" Value="1"/>
   <Enum Name="Busy" Value="2"/>
   <Enum Name="Retry" Value="3"/>
   <Range Name="Good Value" LowValue="4" HighValue="7"/>
   <Default Name="Bad Default"
         ValueError="Bad Default Description"/>
</Enumset>
```

## <DisplayDefaults>

This element defines which fields/bus/signals should be inserted into the Packet Viewer window by default.

**Required** Yes

**Attributes**

| Name | Required | Value | Comment |
|---|---|---|---|
| FieldDirection | no | "LeftToRight" (default) | Specifies a left to right ordering of fields in the header tab of the Packet Viewer window. |
| | | "RightToLeft" | Specifies a right to left ordering of fields in the header tab of the Packet Viewer window. |
| HeaderWidth | no | integer (default: "32") | Specifies the width (in bits) in which to draw the grid within the Header pane of the Packet Viewer. It is useful for specifying the word-size in which to draw individual samples of a packet. |

**Contains**

- "<DisplayField>" on page 77 (required)

**Contained By**

- "<ProtocolFamily>" on page 102

**Example**

```
<DisplayDefaults HeaderWidth="20">
   <DisplayField Name="Sample Number" Width="50"/>
   <DisplayField Name="My Packet" Width="100"/>
</DisplayDefaults>
```

## <DisplayField>

This element names a field that is displayed as a data column in the Packet Viewer window by default, and it specifies the width of the column.

This element defines the characteristics of a field when inserted into the Packet Viewer window. The name of the field can be any existing bus/signal, including other bus/signals generated by the Packet Decoder, like "Direction" or "{Protocol Name} Packet".

There are four columns that are automatically generated by the Packet Decoder tool:

- "Sample Number" — contains the logic analyzer sample number corresponding to the captured data.
- "Time" — contains the logic analyzer time corresponding to the captured data.
- "(ProtocolFamily name) Packet" — contains the main packet type decodes. For example, if the protocol family name is "Ethernet", the main packet type decodes appear in a column named "Ethernet Packet".
- "Direction" — can contain the name of the Packet Decoder tool or a name you specify in the Packet Decoder tool's properties. This is useful when multiple Packet Decoder tools are used for different directions of a serial link and the tools are named to identify the data direction.

**Required** Yes

**Attributes**

| Name | Required | Value | Comment |
|---|---|---|---|
| Name | yes | string | Specifies the name of a field or a bus/signal to be inserted a Packet Viewer window when a new Packet Viewer window is created. |
| Width | no | integer (default: determined by Packet Viewer window) | Specifies the column width in pixels. |

**Contains** None

**Contained By**

- "<DisplayDefaults>" on page 76

**Example**

```
<DisplayDefaults>
   <DisplayField Name="Direction" Width="50"/>
   <DisplayField Name="My Protocol Packet" Width="100"/>
</DisplayDefaults>
```

## <Enum>

This element defines an enumeration that assigns a meaningful string to a value.

**Required** No

**Attributes**

| Name | Required | Value | Comment |
|---|---|---|---|
| Name | yes | string | The name assigned to the value. |
| Value | yes | string | The value the "Name" above will be assigned to. |
| KDChar | no | "Kchar" or "Dchar" (default: "Dchar") | Specifies if the value is a K character or D character. Needed to properly label values in the Lane Data column of the Listing window. |
| ValueError | no | string | If present, then the ValueError attribute specifies that this value is an error and will appear in Red in the Packet Viewer window. A matching string must appear in the <ProtocolErrors> element (see page 101). See "Describing Protocol Errors" on page 44. |
| Description | no | string | Shows a tool tip when the value is present in the Packet Viewer window. |

**Contains** None

**Contained By**

- "<Enumset>" on page 79
- "<Field>" on page 80 (Enum can be added inline to field definitions as a shortcut to creating an enumset external to the protocol. See the example below.)

**Example**

```
<Enum Name="Retry" Value="0" Description="The response was a RETRY"/>
<Enum Name="Error" Value="1" ValueError="Packet Error"/>
```

An example of Enum, Range, and Default added inline to a field definition:

```
<Field Name="XYZ" Length="3">
   <Enum Name="ABC" Value="#b000"/>
   <Enum Name="DEF" Value="#b001"/>
   <Range Name="GHI" LowValue="#b010" HighValue="#b011"/>
   <Default Name="JKL"/>
   ...
</Field>
```

# <Enumset>

This element defines a collection of enumerations that assign meaningful strings to values.

**Required** No

**Attributes**

| Name | Required | Value | Comment |
|---|---|---|---|
| Name | yes | string | The name of the enumset. Field elements that have enumerations will reference enumerations with this name. |

**Contains**

- "<Enum>" on page 78
- "<Range>" on page 103
- "<Default>" on page 75

**Contained By**

- "<ProtocolFamily>" on page 102
- "<SymbolDecode>" on page 107

**Example**

```
<Field Name="Status" Length="2" Enumset="StatusEnum"/>
   ...
   <Enumset Name="StatusEnum">
      <Enum Name="OK" Value="0"/>
      <Enum Name="Error" Value="1"/>
      <Enum Name="Busy" Value="2"/>
      <Enum Name="Retry" Value="3"/>
   </Enumset>
```

# <Field>

This element defines a field.

**Required** Yes

**Attributes**

| Name | Required | Value | Comment |
|---|---|---|---|
| Name | yes | string | Name of the field. |
| Length | yes | formula (see page 55) | A formula specifying the length of the field. This can be a fixed number, or be dependent of the value of another field. The length of a packet must equal the sum of the lengths of all fields. |
| Type | no | "Data" (default) | Normal data field. |
| | | "Protocol" | Decoding will continue using the protocol in the given field name. This is useful for decoding multiple layers of protocol. It is also useful in sharing common decoding definitions between packets. |
| | | "ProtocolField" | Decoding will continue using the protocol given with the name of the enumeration set by the value of the field. This saves the user from having multiple Select formulas to select a new protocol to start decoding from. See "ProtocolField Example" on page 83. |
| | | "Reserved" | A reserved field. |
| | | "Payload" | A Payload field. If there are multiple payload fields, they are concatenated together. The content of the payload field is displayed in the Packet Viewer window's Payload tab. |
| | | "Address" | Specifies that this field contains the starting address for the payload. Only one address field is allowed per packet. If an address field is present in a packet, it will be used as the starting address in the Payload tab of the Packet Viewer window. Only the display is affected by this attribute value – it does not affect the decode. |
| | | "Lookahead" | Used to extract bits later in the packet. This is useful when decoding of a field is dependent upon bits later on in the packet. Use of the BitOffset or AbsoluteBitOffset attributes are required to specify at which bit the data should be extracted. |
| | | "Hidden" | The field will not be displayed in any windows. This is typically used for fields after a <SegmentedField> element (see page 106). |
| | | "Segment" | Like the "Hidden" type, this is typically used for fields after a <SegmentedField> element (see page 106). However, instead of being hidden, the field will be visible in the Header tab of the Packet Viewer window and can be inserted as a column, but it will not be present in the Details tab. |

| Name | Required | Value | Comment |
|---|---|---|---|
| Enumset | no | string | Name of the enumset to be used for a more meaningful display of the data. |
| Enum | no | string | Instead of displaying a value, a more meaningful name will be displayed. This is a shortcut to creating a one element enumset. |
| ProtocolFamily | no | string | Used to specify a multi-layer protocol. The name of the field will be used as a protocol name in the given ProtocolFamily name to continue decoding. |
| AbsoluteBitOffset | no | integer | Offset the number of bits from the beginning of the packet. Normally used with the "Lookahead" type field. |
| BitOffset | no | integer | Offset the number of bits relative to the current position in the packet. |
| Format | no | "Hex", "Decimal", "Octal", "Binary", "Unicode", or "Ascii" | Specify the default base when displaying the field in the Packet Viewer window or Event Editor. |
| ExcludeCRC | no | "T" or "F" | Use of this attribute will remove the field from any CRC calculation. This attribute can exclude a field from a <FieldContainer> element (see page 84). |
| Value | no | integer | Value that the field should be. If the value does not match the field value, then use of the ValueError is needed to specify the error |
| ValueFunction | no | string | Specify a function to calculate the expected value of the field. This is normally used for CRC computation. The syntax is "LibraryName:FunctionName". See "Using ValueFunctions" on page 51. |
| ValueInput | no | string | Specifies the input to be passed to the function specified in ValueFunction. Concatenation of fields to be passed is typically done by wrapping a group of fields into a field container element. The name of the field container would be used for the ValueInput value. |
| Select | no | formula (see page 55) | A formula that is executed to determine if the field is to be used or not for the current packet. |
| Description | no | string | A tool tip string that will display when hovering over the field in the Packet Viewer window. |
| TransformFunction | no | string | Specify a function to transform the value of the field. This would be required when the ordering of the bits/bytes of the field need to be modified. The string value should be in the form: "LibraryName:FunctionName". See "Using TransformFunctions" on page 52. |
| ValueError | no | string | A string specifying the error associated with the value of this field. A matching string must appear in the "<ProtocolErrors>" on page 101 element. See "Describing Protocol Errors" on page 44. |

| Name | Required | Value | Comment |
|---|---|---|---|
| PayloadFormat | no | "BigEndian" (default) | If this field is a payload field, then this specifies the default payload ordering as BigEndian. |
| | | "LittleEndian" | If this field is a payload field, then this specifies the default payload ordering as LittleEndian. |
| EncodeRule | no | formula (see page 55) | Specifies how the entered value when triggering, searching, or filtering on the field should be transformed before performing the operation. See "To display and use full values for partial bit fields" on page 49. |
| DecodeRule | no | formula (see page 55) | Specifies how the value of the field should be transformed prior to display. See "To display and use full values for partial bit fields" on page 49. |
| DecodeString | no | string | Allows C language printf-style formatting to give you great flexibility when decoding data. See "To decode fields with printf-style format strings" on page 49. |
| DisplayLength | no | formula (see page 55) | Specifies the length of the value to display (after the transform has occurred). See "To display and use full values for partial bit fields" on page 49. |

## Contains

Field definitions can contain Enum, Range, and Default elements as a shortcut to creating an Enumset. See the example in "<Enum>" on page 78.

- "<Enum>" on page 78
- "<Range>" on page 103
- "<Default>" on page 75

## Contained By

- "<Protocol>" on page 99
- "<Header>" on page 86
- "<Payload>" on page 98
- "<Trailer>" on page 108
- "<FieldContainer>" on page 84
- "<FieldGroup>" on page 85
- "<MetaField>" on page 89
- "<RepetitiveFields>" on page 104

## Example

```
<Protocol Name="ReadPacket" ProtocolLayer="Transaction">
   <Field Name="ControlBits" Length="3" Enumset="ControlBitsEnum"/>
   <Field Name="ControlBits" Type="ProtocolField"/>
   <Field Name="Data" Length="10"/>
</Protocol>
<Protocol Name="Read Modify Write Packet"
      ProtocolLayer="Transaction">
   <Field Name="Modify Bits" Length="3"/>
   <Field Name="Reserved" Length="2"/>
</Protocol>
<Protocol Name="Read - Flush Packet" ProtocolLayer="Transaction">
```

```
   <Field Name="Flush bits" Length="5"/>
</Protocol>
```

**ProtocolField Example**

The decoder should decode the next part of the packet depending upon the HeaderType field value. One way of doing this is to use Type="ProtocolField". It will use the enumeration value of the "HeaderType" field as the name of the next protocol to decode.

```
<Protocol Name="Header" ProtocolLayer="Data">
   <Field Name="HeaderType" Enumset="HeaderTypes" Length="3"/>
   <Field Name="HeaderType" Type="ProtocolField"/>
</Protocol>

<Enumset Name="HeaderTypes">
   <Enum Name="Read" Value="0"/>
   <Enum Name="Write" Value="1"/>
   <Enum Name="UpdateFC" Value="2"/>
   ...
</Enumset>

<Protocol Name="Read" ProtocolLayer="Data">
...

<Protocol Name="Write" ProtocolLayer="Data">
...

<Protocol Name="UpdateFC" ProtocolLayer="Data">
...
```

Instead of:

```
<Protocol Name="Header" ProtocolLayer="Data">
   <Field Name="HeaderType" Enumset="HeaderTypes" Length="3"/>
   <Field Name="Read" Type="Protocol" Select="HeaderType==0"/>
   <Field Name="Write" Type="Protocol" Select="HeaderType==1"/>
   <Field Name="UpdateFC" Type="Protocol" Select="HeaderType==2"/>
   ...
</Protocol>

<Enumset Name="HeaderTypes">
   <Enum Name="Read" Value="0"/>
   <Enum Name="Write" Value="1"/>
   <Enum Name="UpdateFC" Value="2"/>
   ...
</Enumset>

<Protocol Name="Read" ProtocolLayer="Data">
...

<Protocol Name="Write" ProtocolLayer="Data">
...

<Protocol Name="UpdateFC" ProtocolLayer="Data">
...
```

**See Also**

- "Using Formulas in Field Descriptions" on page 58

## <FieldContainer>

This element defines a logical grouping of fields that will be concatenated for use typically by a ValueInput function.

**Required** No

**Attributes**

| Name | Required | Value | Comment |
|---|---|---|---|
| Name | yes | string | Name used to identify the field container. This name must be referenced by the ValueInput attribute to be used properly. See <Field> element (see page 80). |

**Contains**

- "<Header>" on page 86
- "<Field>" on page 80 (required)
- "<FieldContainer>" on page 84
- "<FieldGroup>" on page 85
- "<MetaField>" on page 89
- "<Payload>" on page 98
- "<RepetitiveFields>" on page 104
- "<SegmentedField>" on page 106
- "<Trailer>" on page 108

**Contained By**

- "<Protocol>" on page 99
- "<Header>" on page 86
- "<Payload>" on page 98
- "<Trailer>" on page 108

**Example**

```
<FieldContainer Name="CRC Data">
   <Header>
      <Field Name="OriginatorID" Length="5"/>
      <Field Name="Length" Length="5"/>
   </Header>
   <Payload>
      <Field Name="Data" Length="Length*8" Type="Payload"/>
   </Payload>
</FieldContainer>
<Trailer>
   <Field Name="CRC" Length="32" ValueFunction="Protocols:CRC32"
         ValueInput="CRC Data"/>
</Trailer>
```

## <FieldGroup>

This element defines a fixed length group of fields where only one field can be selected (with the Select attribute).

**Required** No

**Attributes**

| Name | Required | Value | Comment |
|---|---|---|---|
| Name | yes | string | Name of the field group to be used |
| Length | yes | formula (see page 55) | Specifies the length of the fields. A formula value of '#VARIABLE' can be used to specify that, of the fields contained in the field group, the length is variable. |

**Contains**

- “<Field>" on page 80 (required)
- “<FieldGroup>" on page 85
- “<MetaField>" on page 89
- “<SegmentedField>" on page 106

**Contained By**

- “<Protocol>" on page 99
- “<Header>" on page 86
- “<Payload>" on page 98
- “<Trailer>" on page 108
- “<FieldContainer>" on page 84
- “<FieldGroup>" on page 85

**Example**

```
<Field Name="Words" Length="10"/>
<FieldGroup Name="Payload" Length="'#VARIABLE'">
   <Field Name="Payload" Length="Words*32" Type="Payload"
         Select="Words!=#h0"/>
   <Field Name="Payload" Length="1024*32" Type="Payload"
         Select="Words==#h0"/>
</FieldGroup>
```

In this example, if the contents of the Words field is zero, the length of the Payload field is 1024*32 bits; otherwise, the number of bits in the Payload field is the number in the Words field multiplied by 32.

## <Header>

This element defines a logical grouping of fields.

The <Header> element is not required. Fields can be assumed to be in the header because the <Payload> element is required. This element is a convenience for organizing fields in the protocol description file.

The deepest header decoded in a packet is the one that is displayed in the Packet Viewer window's Header tab.

**Required** No

**Attributes** No attributes defined.

**Contains**

- "<Field>" on page 80 (required)
- "<FieldContainer>" on page 84
- "<FieldGroup>" on page 85
- "<MetaField>" on page 89
- "<RepetitiveFields>" on page 104
- "<SegmentedField>" on page 106

**Contained By**

- "<Protocol>" on page 99
- "<FieldContainer>" on page 84

**Example**

```
<Protocol Name="Write Packet" ProtocolLayer="Data">
   <Header>
      <Field Name="OriginatorID" Length="5"/>
      <Field Name="Length" Length="5"/>
   </Header>
   <Payload>
      <Field Name="Data" Length="Length*8" Type="Payload"/>
   </Payload>
</Protocol>
```

## <Label>

This element describes a bus/signal that is required to perform protocol decode. Included are attributes that specify formulas about how SOP/EOP/Valid and selection are performed.

**Required** Yes

**Attributes**

| Name | Required | Value | Comment |
|---|---|---|---|
| Name | yes | string | Name of the bus/signal that is required. |
| Width | yes | integer | Required number of signals the bus/signal must have. |
| Type | yes | "8bData" | The label contains *8bdata* (see page 113). |
| | | "10bData" | The label contains *10bdata* (see page 113). |
| | | "Data" | The label contains data that will potentially be included in a packet. Typically, there is one label with Type="Data", unless you are decoding a multi-lane bus. |
| | | "Valid" | The label contains data pertaining to the validity of the data. Use these labels in conjunction with data label's Valid attribute. See "Labels that Identify Valid Data" on page 29. |
| | | "8b/10b" | The label is used to switch between *8bdata* (see page 113) and *10bdata* (see page 113). |
| | | "K/D" | The label is used to specify when *K characters* (see page 113) are present. |
| | | "Bonded" | The label is used to specify when multi-lane data is bonded (another form of valid). Samples are automatically filtered by the signals identified. |
| | | "Idle" | The label is used to specify when the data is not-valid due to being Idle. Samples are automatically filtered by the signals identified. |
| | | "Frame" | The label is used to specify framing (SOP, EOP). |
| | | "MetaData" | The label is used to contain more than one symbol's worth of data that should be partitioned into individual symbols. |
| | | "Status" | The label contains general status that is used in a data label's formulas. |
| Sop | yes | formula (see page 55) | A formula used to specify when a new packet is starting. This attribute can be in any <Label> element that has a Type="Data", Type="10bData", Type="8bData", or Type="MetaData" attribute. |
| Lane | no | integer | Specifies which lane this label is used for. |
| Meta | no | string (label_or_constant\#bits, label_or_constant\#bits, ...) | Lets you add additional information to a packet. See "To add information to a packet" on page 50. |

| Name | Required | Value | Comment |
|---|---|---|---|
| Select | no | formula (see page 55) | A formula used to conditionally enable or disable the use of the label for other formulas. If no formula is present, then the label will always be selected. |
| Valid | no | formula (see page 55) | A formula used to specify when the data for the label should be included as part of a packet. If no formula is present, then all data will assumed to be valid. |
| Eop | no | formula (see page 55) | A formula used to specify when a packet is ending. If no formula is present, then no end-of-packet will be specified. Each packet will span from the start-of-packet sample to the next start-of-packet sample. |
| OrderedSetSop | no | formula (see page 55) | A formula used to specify when a packet is an *ordered set* (see page 113). If no formula is present, then no packets will be specified as ordered sets. See Chapter 5, “Multi-Lane Serial Link Concepts,” starting on page 67. |
| Kchar | no | formula (see page 55) | A formula used to specify when the data is a *K character* (see page 113) or not. If this formula is not present, then all data is assumed to be *D characters* (see page 113). |
| PacketData | no | formula (see page 55) | A formula used to specify which states should be added to the packet. If no formula is present, then all states between SOP and EOP or SOP + Packet Length will be added to the packet unless idle or bonded formulas specify otherwise. |
| Value | no | formula (see page 55) | A formula used to specify additional operations on the data provided by the label. Examples would be bitwise shifting or bitwise masking. |

## Contains

None

## Contained By

- “<Bus>" on page 72

## Example

```
<Bus Name="Utopia">
   <Label Name="DATA" Width="16" Type="Data" Sop="'SOP'==#b1"
         Eop="'EOP'==#b1" Valid="'CLK'==#b1" />
   <Label Name="SOP" Width="1" Type="Frame"/>
   <Label Name="EOP" Width="1" Type="Frame"/>
   <Label Name="CLK" Width="1" Type="Valid"/>
   <BusProtocol Name="IEEE 802.3 (Ethernet V2)" Type="Packet"/>
</Bus>
```

## See Also

- “Labels that Contain Data" on page 28
- “Labels that Identify Valid Data" on page 29
- “If Your Serial Bus Has Lanes" on page 30
- Chapter 5, “Multi-Lane Serial Link Concepts,” starting on page 67
- “Using Formulas in Bus/Signal Label Descriptions" on page 56

## <MetaField>

This element defines a field that has subfields where you may want to view the constituent parts. This creates a tree-structure in the Details tab of the Packet Viewer window.

**Required** No

**Attributes** Same as the "<Field>" on page 80 element.

**Contains**

- "<Field>" on page 80 (required)
- "<MetaField>" on page 89
- "<SegmentedField>" on page 106

**Contained By**

- "<Protocol>" on page 99
- "<Header>" on page 86
- "<Payload>" on page 98
- "<Trailer>" on page 108
- "<FieldContainer>" on page 84
- "<FieldGroup>" on page 85
- "<MetaField>" on page 89
- "<RepetitiveFields>" on page 104

**Example**

```
<MetaField Name="TransactionID" Length="12">
   <Field Name="Sequence Number" Length="7" />
   <Field Name="Originator ID" Length="5" />
</MetaField>
```

## <PacketDisplay>

This element defines the default display of a packet in the Packet Viewer window. Additionally, tool tip text can be specified to give additional information about the packet type.

**Required** No

**Attributes**

| Name | Required | Value | Comment |
|---|---|---|---|
| ForegroundColor | no | color (see page 46)<br>(default: "Black") | Foreground color of the packet in the Packet Viewer window. |
| BackgroundColor | no | color (see page 46)<br>(default: "White") | Background color of the packet in the Packet Viewer window. |
| Description | no | string | Text to be displayed in tool tips for the packet in the Packet Viewer window. |

**Contains** None

**Contained By**

- "<PacketType>" on page 95

**Example**

```
<PacketType Name="Write">
   <PacketMask Width="3" Value="2"/>
   <PacketDisplay ForegroundColor="Pink" BackgroundColor="Yellow"
         Description="This is a write packet. Write packets require
         a response packet to acknowledge the receipt of data."/>
</PacketType>
```

## <PacketHighlightRule>

This element defines a single packet highlighting rule to apply to the selected packet within the Lanes tab of the Packet Viewer window. Recall that these rules are applied in order from top to bottom. Therefore, latter rules will override earlier rules for cells that match more than one rule.

A rule contains one or more optional attributes that define whether the rule applies to a particular cell within the selected packet. Rule attributes can refer to either a symbol, a decoded field, a segment of a packet, a layer in a packet, or an entire protocol. Therefore, rules can be applied at different levels of granularity to the selected packet. Since rules are applied in order, more general rules should be defined earlier in the list compared to more specific rules in order to achieve the correct packet highlighting effect. Furthermore, if more than one attribute is defined within a rule, then a cell must match all the defined attributes for the highlighting rule to apply. In other words, attributes are AND'ed together rather than OR'ed. For example, consider the following rule:

```
<PacketHighlightRule FieldName="LCRC" ForegroundColor="Black"
      BackgroundColor="Gray" DisplayName="LCRC"/>
```

The above rule will highlight every cell with black text on a gray background whose decoded field name is "LCRC" and whose decoded field type is "CRC".

Last, note that if a rule only partially overlaps a cell within the Lanes tab, then the entire cell is considered as belonging to that rule (that is, there are no partial matches for a cell). Therefore, if only a single bit occurs within a cell for a particular rule, the entire cell will be highlighted according to that rule. This can have interesting side effects. For example, if a cell spans two or more decoded fields and there exists rules for each of these decoded fields, then the last defined rule in the list will apply as the highlight for that cell.

Finally, the Lane8bValue and LaneKDChar attributes can only be combined with one another. If other attributes are combined with them, they are simply ignored. The other attributes can be combined with one another in any possible combination.

**Required** No

**Input Attributes**

| Name | Required | Value | Comment |
|---|---|---|---|
| Lane8bValue | no | integer | Numeric value for a particular symbol. Example: "#hfb". May only be combined with the LaneKDChar attribute. |
| FieldName | no | string | Decoded field name. |
| FieldType | no | string (see the Type attribute values in "<Field>" on page 80) | Decoded field type. |
| LaneKDChar | no | "Kchar", "Dchar", or "DontCare" | Character type for a particular symbol. May only be combined with the Lane8bValue attribute. |
| PacketSegment | no | "Header", "Payload", "Trailer", or "Any" | Packet segment type. |
| ProtocolLayer | no | string | Protocol layer name. |
| ProtocolFamily | no | string | Protocol family name. |

## Output Attributes

| Name | Required | Value | Comment |
|---|---|---|---|
| BackgroundColor | no | color (see page 46) (default: packet type background color) | The cell's background color. |
| ForegroundColor | no | color (see page 46) (default: packet type foreground color) | The cell's foreground color. |
| DisplayName | no | string (default: symbolic decode string) | The text displayed in the cell when the Lane tab's **Field Decode** option is selected. |

## Contains

None

## Contained By

- "<PacketHighlightRules>" on page 93

## Example

```
<PacketHighlightRules>
   <PacketHighlightRule PacketSegment="Header"
         ForegroundColor="Black" BackgroundColor="LightYellow"
         DisplayName="Header"/>
   <PacketHighlightRule LaneKDChar="KChar" ForegroundColor="Black"
         BackgroundColor="Green"/>
</PacketHighlightRules>
```

## <PacketHighlightRules>

This element defines how cell highlighting should be applied in the Lanes tab of the Packet Viewer window. Each cell within the Lanes tab represents a single symbol of the selected packet and can be highlighted according to various rules defined in this element. By default, a cell's color matches the color of the corresponding packet. Therefore, if no rules are defined, then cells will appear as the color of the selected packet. Otherwise, the rules defined within this element allow customized highlighting and display of particular fields and symbols within the selected packet.

The nested rules within this element define how each cell of the selected packet can be highlighted with a foreground color, background color, and a display name. Each defined <PacketHighlightRule> is applied *in order* to the selected packet. Therefore, rules that appear later in the list will override earlier rules if they apply to the same cell. Essentially, each rule is applied in order to the selected packet until all rules have been applied. The resulting highlighting is then displayed within the Lanes tab of the PacketViewer.

Last, these rules only apply to regular packets and not to ordered sets. Ordered sets will always appear as the color of the ordered set.

**Required** No

**Attributes** No attributes defined.

**Contains**

- "<PacketHighlightRule>" on page 91

**Contained By**

- "<ProtocolFamily>" on page 102

**Example**

```
<PacketHighlightRules>
   <PacketHighlightRule PacketSegment="Header"
         ForegroundColor="Black" BackgroundColor="LightYellow"/>
   <PacketHighlightRule FieldName="Payload" FieldType="Payload"
         ForegroundColor="White" BackgroundColor="DarkRed"/>
</PacketHighlightRules>
```

## <PacketMask>

This element defines a bit mask to be used to pattern match a packet. The bit mask created by this element will be used to determine the packet type during the decoding operation. Note that the presence of multiple packet masks will be concatenated together to create one large mask.

If the bits that identify the packet type do not appear at the start of the packet, you can use the BitOffset attribute to specify their offset from the SOP.

### Required

Yes

### Attributes

| Name | Required | Value | Comment |
|---|---|---|---|
| Width | yes | integer | The width of the mask to be created. Currently there is a limitation of 32 bits per mask. |
| BitOffset | no | integer (default: "0") | Number of bits from the start-of-packet the width/value should be offset. |
| Value | no | integer | The value within the mask. |

### Contains

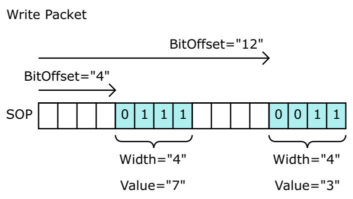
None

### Contained By

- "<PacketType>" on page 95

### Example

```
<PacketType Name="Write Packet">
   <PacketMask Width="4" BitOffset="4" Value="7"/>
   <PacketMask Width="4" BitOffset="12" Value="3"/>
</PacketType>
```



## <PacketType>

This element defines a packet type. The name of the packet type will appear in the Packet Viewer for the packet summary when a packet data matches the specified mask under the PacketMask element. The default colorization of the packet is specified under the PacketDisplay element.

**Required** Yes

**Attributes**

| Name | Required | Value | Comment |
|---|---|---|---|
| Name | yes | string | Name of the packet. |
| Any | no | "T" | "Any" packets let you search, filter, or trigger on general packet types (using the Event Editor), rather than on specific ones. |
| | | "F" (default) | This packet type is not included in the "Any" packets group. |
| Encodeable | no | "T" (default) | True, if this packet should be considered as a triggerable packet. |
| | | "F" | False, if this packet should not be considered as a triggerable event. Sometimes certain packet types cannot be triggered on, and using this attribute can turn off these packets from appearing in the Event Editor. |
| Protocol | no | string | The protocol to be used to encode the packet for the Event Editor. Each PacketType can specify its own Protocol attribute, which will override any specified by the PacketTypeGroup or PacketTypes elements. |

**Contains**

- “<PacketMask>" on page 94 (required)
- “<PacketDisplay>" on page 90

**Contained By**

- “<PacketTypes>" on page 97
- “<PacketTypeGroup>" on page 96

**Example**

```
<PacketType Name="Any Write Packet" Decodeable="F">
   <PacketMask Width="1" BitOffset="5" Value="1"/>
</PacketType>
<PacketType Name="Write Request Packet">
   <PacketMask Width="1" BitOffset="5" Value="1"/>
   <PacketMask Width="1" BitOffset="9" Value="1"/>
</PacketType>
```

## <PacketTypeGroup>

This element specifies a logical grouping of packet types. Each PacketTypeGroup will appear in the Event Editor as a folder. The PacketTypeGroup can also specify a protocol that will be used by the Event Editor and Packet Decoder to decode the packets.

**Required** No

**Attributes**

| Name | Required | Value | Comment |
|---|---|---|---|
| Name | yes | string | Name for the folder in the Event Editor |
| Protocol | no | string | Name of the protocol to use by the Event Editor and Packet Decoder to decode packets. |

**Contains**

- "<PacketTypeGroup>" on page 96
- "<PacketType>" on page 95 (required)

**Contained By**

- "<PacketTypes>" on page 97
- "<PacketTypeGroup>" on page 96

**Example**

```
<PacketTypes Name="My Packets" Default="Packet Type 1"
      Protocol="My Packets">
   <PacketTypeGroup Name="Write Packets" Protocol="Write Packets">
      ...
   </PacketTypeGroup>
</PacketTypes>
```

## <PacketTypes>

This element specifies the main packet types that will be used to do multiple things:

- First, each PacketType entry will appear in the Event Editor interface as a starting point for setting up a trigger. PacketTypeGroups will appear as folders in the Event Editor interface.
- Also, each packet type will be considered as a possible match in determining which string to display in the Packet Viewer window for a packet.

Packet types are searched in the order that they appear in the protocol description file.

**Required** Yes

**Attributes**

| Name | Required | Value | Comment |
|---|---|---|---|
| Name | yes | string | Name of the packet types. |
| Protocol | no | string | The protocol to be used to decode/encode the packet. Each PacketTypeGroup and PacketType can specify its own Protocol attribute, which will override this one. |
| Default | no | string (default: the first packet type) | The Event Editor default packet type among the PacketType definitions that follow. |

**Contains**

- "<PacketTypeGroup>" on page 96
- "<PacketType>" on page 95 (required)

**Contained By**

- "<ProtocolFamily>" on page 102

**Example**

```
<PacketTypes Name="My Protocol Packets" Protocol="My Protocol"
      Default="Packet Type 1">
   <PacketTypeGroup Name="Read Packet Types">
      <PacketType Name="Packet Type 1">
         <PacketMask Width="1" BitOffset="3" Value="0"/>
      </PacketType>
   </PacketTypeGroup>
   <PacketTypeGroup Name="WritePacketTypes">
      <PacketType Name="Packet Type 2">
         <PacketMask Width="1" BitOffset="3" Value="1"/>
      </PacketType>
   </PacketTypeGroup>
</PacketTypes>
```

## <Payload>

This element defines a logical grouping of fields that comprise the payload portion of a protocol layer.

**Required** No

**Attributes** No attributes defined.

**Contains**

- "<Field>" on page 80 (required)
- "<FieldContainer>" on page 84
- "<FieldGroup>" on page 85
- "<MetaField>" on page 89
- "<RepetitiveFields>" on page 104
- "<SegmentedField>" on page 106

**Contained By**

- "<Protocol>" on page 99
- "<FieldContainer>" on page 84

**Example**

```
<Payload>
   <Field Name="Data" Length="128" Type="Payload"/>
</Payload >
```

## <Protocol>

This element defines a logical structure of protocol.

**Required** Yes

**Attributes**

| Name | Required | Value | Comment |
|---|---|---|---|
| Name | yes | string | Name of the protocol referenced by other parts of the file. |
| ProtocolLayer | yes | string | Meaningful name to specify the layer. This name is used in the Details tab of the Packet Viewer window and the Event Editor dialog to organize fields into logical groups. |
| Type | No | "Lane" | The protocol is used for lane or *ordered set* (see page 113) decoding. |
|  |  | "Packet" (default) | The protocol is used for packet decoding. |

**Contains**

- “<Header>" on page 86
- “<Field>" on page 80 (required)
- “<FieldContainer>" on page 84
- “<FieldGroup>" on page 85
- “<MetaField>" on page 89
- “<Payload>" on page 98
- “<RepetitiveFields>" on page 104
- “<SegmentedField>" on page 106
- “<Trailer>" on page 108

**Contained By**

- “<ProtocolFamily>" on page 102

**Example**

```
<Protocol Name="Write Packet" ProtocolLayer="Data">
   <Field Name="OriginatorID" Length="5"/>
   <Field Name="Data" Length="5"/>
</Protocol>
```

## <ProtocolError>

This element defines a protocol error that is referenced by enums and fields.

**Required** Yes

**Attributes**

| Name | Required | Value | Comment |
|---|---|---|---|
| Name | yes | string | The name of the error which must match exactly with the fields and enums that use it. |
| Description | yes | string | The tool tip text to be used when hovering over a packet with an error. |

**Contains** None

**Contained By**

- "<ProtocolErrors>" on page 101

**Example**

```
<ProtocolErrors>
   <ProtocolError Name="CRC Error" Description="Bad CRC"/>
   <ProtocolError Name="Bad Status Value"
         Description="Bad Status Value"/>
</ProtocolErrors>
```

## <ProtocolErrors>

This element defines the protocol errors specified by the protocol that are possible during packet decode.

| | |
|---|---|
| **Required** | No |
| **Attributes** | No attributes defined. |
| **Contains** | · "<ProtocolError>" on page 100 (required) |
| **Contained By** | · "<ProtocolFamily>" on page 102 |

**Example**

```
<ProtocolErrors>
   <ProtocolError Name="CRC Error" Description="Bad CRC"/>
   <ProtocolError Name="Bad Status Value"
         Description="Bad Status Value"/>
</ProtocolErrors>
```

## <ProtocolFamily>

This is the root element of the XML document. All other elements must be contained within this element.

If you want to license your protocol description, use the LicenseName, LicenseVendor, and LicenseVersion attributes. When these attributes are used, a license must be found before the protocol description can be loaded into the *Keysight Logic Analyzer* application. License creation tools are part of the standard FLEXlm developer's kit.

**Required** Yes

**Attributes**

| Name | Required | Value | Comment |
|---|---|---|---|
| Name | yes | string | Specifies the name that will appear in the Packet Decoder list of protocols. |
| Version | yes | decimal | Must be 1.1 for software version 3.65. Must be 1.0 for previous versions. |
| LicenseName | no | string | FlexLM License Name |
| LicenseVendor | no | string | FlexLM License Vendor |
| LicenseVersion | no | decimal | FlexLM License Version |
| TransmissionOrder | no | "LSBFirst" | Least significant bits are transmitted first in the packet. See "Byte/Bit Order Requirements" on page 18. |
| | | "MSBFirst" (default) | Most significant bits are transmitted first in the packet. |

**Contains**

- "<Bus>" on page 72 (required)
- "<PacketTypes>" on page 97 (required)
- "<Enumset>" on page 79
- "<SymbolDecode>" on page 107
- "<Protocol>" on page 99 (required)
- "<DisplayDefaults>" on page 76 (required)
- "<PacketHighlightRules>" on page 93
- "<ProtocolErrors>" on page 101

**Contained By** None (this is the root element)

**Example**

```
<ProtocolFamily Name="My Protocol" Version="1.1"
      LicenseName="MyProtocol_License"
      LicenseVendor="My Company" LicenseVersion="1.1">
   ...
</ProtocolFamily>
```

**See Also**

- "Step 5: Choose a unique protocol family name" on page 26

## <Range>

This element assigns a meaningful string to a range of values.

**Required** No

**Attributes**

| Name | Required | Value | Comment |
|---|---|---|---|
| Name | yes | string | The name assigned to the value range. |
| LowValue | yes | string | The low value of the range. |
| HighValue | yes | string | The high value of the range. |
| Description | no | string | A description of the range. |
| ValueError | no | string | If present, the ValueError attribute specifies that this range of values is an error and will appear in Red in the Packet Viewer window. A matching string must appear in the “<ProtocolErrors>" on page 101 element. See “Describing Protocol Errors" on page 44. |
| KDChar | no | "Kchar", "Dchar", or "DontCare" | Specifies if the range of values are K characters or D characters. |

**Contains** None

**Contained By**

- “<Enumset>" on page 79
- “<Field>" on page 80 (Range can be added inline to field definitions as a shortcut to creating an enumset external to the protocol. See the example in “<Enum>" on page 78.)

**Example**

```
<Range Name="Good Value" LowValue="0" HighValue="7"/>
<Range Name="Bad Value" LowValue="8" HighValue="15"
      ValueError="Bad Value Description"/>
```

## <RepetitiveFields>

This element defines a grouping of fields that will repeat x/y times where, x is the overall length specified by the repetitive field element and y is the sum of each field length contained within the repetitive field element.

**Required** No

**Attributes**

| Name | Required | Value | Comment |
|---|---|---|---|
| Length | yes | formula (see page 55) | Length of the overall grouping of fields |
| Hierarchical | no | "T" (default) | True, the name of the repetitive field becomes a layer in the Details tab and each occurrence is indexed (with square brackets) as a child to the layer |
| | | "F" | False, repetitive fields are displayed in the Details tab without any hierarchy. |
| Name | no | string | Name of the Repetitive Field |
| Select | no | formula (see page 55) | A formula that is executed to determine if the fields are to be used or not for the current packet. |

**Contains**

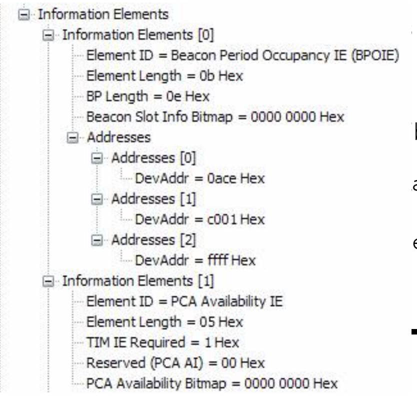
- "<Field>" on page 80 (required)
- "<MetaField>" on page 89
- "<SegmentedField>" on page 106

**Contained By**

- "<Protocol>" on page 99
- "<Header>" on page 86
- "<Payload>" on page 98
- "<Trailer>" on page 108
- "<FieldContainer>" on page 84

**Example**

```
<RepetitiveFields Name="TransactionID" Length="120"/>
   <Field Name="PortRegister" Length="10"/>
   <Field Name="OffsetRegister" Length="10" />
</RepetitiveFields>
```



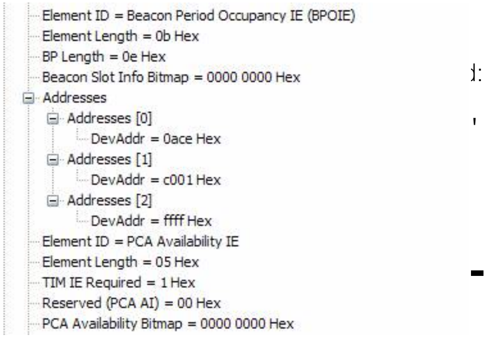
```
Information Elements
  Information Elements [0]
    Element ID = Beacon Period Occupancy IE (BPOIE)
    Element Length = 0b Hex
    BP Length = 0e Hex
    Beacon Slot Info Bitmap = 0000 0000 Hex
    Addresses
      Addresses [0]
        DevAddr = 0ace Hex
      Addresses [1]
        DevAddr = c001 Hex
      Addresses [2]
        DevAddr = ffff Hex
  Information Elements [1]
    Element ID = PCA Availability IE
    Element Length = 05 Hex
    TIM IE Required = 1 Hex
    Reserved (PCA AI) = 00 Hex
    PCA Availability Bitmap = 0000 0000 Hex
```

will each appear 6 times. Each field have a [] appended
in the Packet Viewer window. (For example:

bute wher

ation E

e="Prot



```
Element ID = Beacon Period Occupancy IE (BPOIE)
Element Length = 0b Hex
BP Length = 0e Hex
Beacon Slot Info Bitmap = 0000 0000 Hex
Addresses
  Addresses [0]
    DevAddr = 0ace Hex
  Addresses [1]
    DevAddr = c001 Hex
  Addresses [2]
    DevAddr = ffff Hex
Element ID = PCA Availability IE
Element Length = 05 Hex
TIM IE Required = 1 Hex
Reserved (PCA AI) = 00 Hex
PCA Availability Bitmap = 0000 0000 Hex
```

## <Segment>

This element describes individual segments within a segmented field.

<Segment> elements do not increment the internal bit counter.

**Required** No

**Attributes**

| Name | Required | Value | Comment |
|---|---|---|---|
| Length | yes | integer | The number of bits in the segment. |
| BitOffset | no | integer (default: "0") | Number of bits from the current internal bit offset location the length should be offset. |

**Contains** None

**Contained By**

- "<SegmentedField>" on page 106

**Example**

```
<SegmentedField Name="FrameID" Length="11" Format="Decimal">
   <Segment BitOffset="0" Length="3" />
   <Segment BitOffset="5" Length="8" />
</SegmentedField>
<!-- Now advance the internal bit offset: -->
<Field Name="Frame ID part 1" Length="3" Type="Segment"/>
<Field Name="BSS1"            Length="2" Type="Segment" Value="#b10"/>
<Field Name="Frame ID part 2" Length="8" Type="Segment"/>
```

## <SegmentedField>

This element describes a field that has non-contiguous bits. This can be useful, for example, when clock recovery bits appear periodically within a packet and are not related to the packet's data.

<SegmentedField> elements do not increment the internal bit counter, so the use of segment-type or hidden-type fields (to increment the bit counter) after a segmented field is likely. Hidden-type fields are not displayed. Segment-type fields are displayed in the Header tab but not the Details tab.

**Required** No

**Attributes** Same as the "<Field>" on page 80 element.

**Contains**

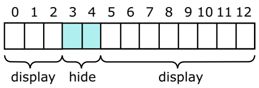
- "<Segment>" on page 105

**Contained By**

- "<Protocol>" on page 99
- "<Header>" on page 86
- "<Payload>" on page 98
- "<Trailer>" on page 108
- "<FieldContainer>" on page 84
- "<FieldGroup>" on page 85
- "<MetaField>" on page 89
- "<RepetitiveFields>" on page 104

**Example**

```
<SegmentedField Name="FrameID" Length="11" Format="Decimal">
   <Segment BitOffset="0" Length="3" />
   <Segment BitOffset="5" Length="8" />
</SegmentedField>
<!-- Now advance the internal bit offset: -->
<Field Name="Frame ID part 1" Length="3" Type="Segment"/>
<Field Name="BSS1"            Length="2" Type="Segment" Value="#b10"/>
<Field Name="Frame ID part 2" Length="8" Type="Segment"/>
```



## <SymbolDecode>

This element defines how *symbol decode* (see page 113) should be performed.

**Required** No

**Attributes** No attributes defined.

**Contains**

- "<Enumset>" on page 79 (required)

**Contained By**

- "<ProtocolFamily>" on page 102

**Example**

```
<SymbolDecode>
   <Enumset Name="Symbol Decode">
      <Enum Name="COM" Value="#hbc" KDChar="Kchar"/>
      <Enum Name="STP" Value="#hfb" KDChar="Kchar"/>
      <Enum Name="TS1" Value="#h45" KDChar="Dchar"/>
   </Enumset>
</SymbolDecode>
```

# <Trailer>

This element defines a logical grouping of fields.

**Required** No

**Attributes** No attributes defined.

**Contains**

- "<Field>" on page 80 (required)
- "<FieldContainer>" on page 84
- "<FieldGroup>" on page 85
- "<MetaField>" on page 89
- "<RepetitiveFields>" on page 104
- "<SegmentedField>" on page 106

**Contained By**

- "<Protocol>" on page 99
- "<FieldContainer>" on page 84

**Example**

```
<Trailer>
   <Field Name="CRC" Length="10" />
</Trailer >
```

# 7 Formula Reference

Formulas are a simple meta-language that are used to make the execution of decoding and framing more dynamic. The use of a formula allows the decoding and framing logic to be dependent upon the current values of bus/signals and packet data.

Formulas are specified in attributes of particular tags using quotes to delineate the contents. Inside the quotes, there are a variety of operands and operators that can be used to provide the dynamic nature of framing and decoding.

The basic structure of a formula contains one or more operands and one or more operators.

[unaryOperator (see page 110)] operand (see page 111) [Operator (see page 110)operand (see page 111)]

## Operators

The list of allowed operators and their precedence (bottom to top) are as follows:

| Operator | Type | Meaning |
|---|---|---|
| \|\| | Binary | Boolean OR |
| .land. | Binary | Boolean AND |
| \| | Binary | Bitwise OR |
| ^ | Binary | Bitwise XOR |
| .band. | Binary | Bitwise AND |
| ==, != | Binary | Equivalence, Not Equivalence |
| .lt., .gt., .le., .ge. | Binary | Less Than, Greater Than, Less Or Equal, Greater Or Equal |
| .lshift.<br>.rshift. | Binary | Left Shift,<br>Right Shift |
| +, - | Binary | Plus, minus |
| *, /, % | Binary | Multiple, Divide, Modulo |
| (,) | Binary | Parentheses |
| +,-,!,~ | Unary | Positive, Negative, Not, Complement |

## Operands

Names of operands can be constant numbers or references to fields or bus/signals.

- "Constants" on page 111
- "Field and Bus/Signal Operand Names" on page 111
- "Ranging" on page 112

### Constants

For constant numbers, the format can be a plain number, which defaults to decimal. Or a specified base can be used to prefix the number.

| Character Prefix | Base | Example |
|---|---|---|
| #b | Binary | Value==#b1 |
| #d | Decimal | Value==#d3 |
| #c | Octal | Value==#c6 |
| #h | Hex | Value==#ha |
| None | Decimal | Value==10 |

**Special #PACKET_LENGTH Constant**

The special '#PACKET_LENGTH' constant returns the length of the framed packet in bits. See "To get the length of variable-length packets" on page 58.

**Special #VARIABLE Constant**

The special '#VARIABLE' constant is used in the Length attribute of the <FieldGroup> element to specify that, of the fields contained in the field group, the length is variable. See "<FieldGroup>" on page 85.

### Field and Bus/Signal Operand Names

Valid characters for operand names:

- Lowercase letters: a-z
- Uppercase letters: A-Z
- Underscore: _
- Period: .
- OpenBracket: [
- CloseBracket: ]
- Colon: :
- Digits: 0-9

All field operands must begin with an upper/lowercase letter and can contain any number of digits and letters or brackets. Spaces may be included, but the operand must be enclosed in single quotes. If the operand contains square brackets, then single quotes are also required.

**Example**

```
TestName==#h1
```

```
'Test Name'==#h1
```

```
'Address[31:2]'==#h1
```

### Ranging

Particular values of a bus/signal or field can be extracted for formula execution through the use of ranges. Ranges are specified with square brackets as follows:

```
Address[31:2]==10
```

This means that 30 bits from the address field will be used in the formula calculation. To specify the field 'Address[31:2]', then quotes must surround the field name and square brackets.

**NOTE** **The name of fields and labels generally has fewer restrictions than the names required for formulas. It may be that the name of the field cannot be used in the formula until the name of the field is changed to be compliant with the formula operand restrictions.**

### Look Around

Formulas can retrieve data from previous or future samples by using braces as follows:

```
Bus{-1}==#h1
```

**NOTE** **Look around in conjunction with ranging is not supported.**

**See Also**

- "To look around" on page 56

# 8 Glossary

8

**8B/10B encoding**: A block coding scheme for high-speed serial and fiber-optic communication links that translates a block of data into a longer block of data that has more transitions between 1's and 0's. The 8B/10B block code maps every byte (8 bits) into a 10-bit value (symbol) that has 3-8 transitions and a balanced number of 1's and 0's. (The 8B/10B block code was designed by IBM in the mid-1980's and has been used in FibreChannel communication links between computers and mass storage devices.)

D

**D character:** In 8B/10B coding, the 10-bit codes for the 256 8-bit values are often referred to using "D" character names that come from the first 5 bits of the 8-bit value separated from the last 3 bits. For example:

D28.2 010 11100

D28.2 represents the encoding for the binary value above, where 28 is the decimal representation of the first 5 bits, and 2 is decimal representation of the last 3 bits.

K

**K character:** In 8B/10B coding, in addition to the 10-bit codes for the 256 8-bit values, there are a few extra 10-bit codes called special characters. Special characters are used for data delimiters like start-of-packet, end-of-packet, idle, and configuration messages.

These special characters are often described with character names, but they use a "K" character instead of a "D". The special characters are:

K28.0 K28.1 K28.2 K28.3 K28.4 K28.5 K28.6 K28.7 K23.7 K27.7 K29.7 K30.7

L

**lane:** Describes one serial channel in situations where multiple serial channels are bonded to transmit greater amounts of data.

O

**ordered set**: In 8B/10B, this refers to a specific combination of characters (symbols). For example, a start-of-frame ordered set might be K25.8 D21.5 D22.2 D22.2.

S

**symbol decoding**: Usually in reference to 8B/10B encoding, symbol decoding describes how 8-bit values map to 10B character names and symbols.

# Index