PROTOCOL INSIGHT

Falcon Trace Validation Builder User Manual

Contact Protocol Insight at:

sales@protocolinsight.com
support@protocolinsight.com
www.protocolinsight.com

The Protocol Insight® Falcon Trace Validation™ Builder tool allows users to create trace validation (TV) analysis tests which can be executed with the Falcon G300 and G400 Series Exerciser/Analyzers.

Trace Validation tests analyze UFS and UniPro traces, performing protocol sequence and packet inspection rule checking based on the nature of the traffic. Trace Validation uses a unique protocol-aware rule checker engine (a state machine) to validate captured traces against the JEDEC CTS and JESD220B/C/2 specification and the MIPI UniPro v1.x CTS and v1.61 specifications to identify protocol errors and failures.

Contents

Trace Validation Builder Overview

The Protocol Insight® Trace Validation Builder tool allows users to create, view and edit trace validation analysis tests which are used by the Falcon Series Exerciser/Analyzer. Trace Validation tests, saved in XML format, are loaded into Falcon Series Exerciser/Analyzer and used to validate UFS and UniPro traces. This provides the user a means to ensure that the format and sequence of the data in a trace is valid and that the tested device performs as expected.

Constructing a Trace Validation test case consists of the following steps:

1. Create a new TV test using the provided template or open a pre-defined test from the Falcon software library.
2. Edit the State Machine to contain one Initial State and two Final States (Success and Failed), at least one Event with one or more Validators, and appropriate Transitions between States.
3. Create the Trace Validation XML file generated by Trace Validation Builder by saving the State Machine File.

Trace Validation Basics

Creating Your Own Trace Validation Tests

From an existing Trace Validation test in the library

The pre-existing Trace Validation tests provided in the Falcon software library can be used as a starting point for new tests since they can be copied and modified. Simply open an existing *.sm (state machine) TV test file from C:\Program Files\Protocol Insight\Falcon\Data\Measurement using Trace Validation Builder, modify it and save a copy with a new name to a subfolder in "C:\Users\[user name]\Documents\Protocol Insight\Falcon\CustomTests".

Important: Do not modify the provided Trace Validation files without first saving a copy to your CustomTests folder. Doing so will overwrite the tests in the software library and require that they be restored to their original state.

Important: The name of Trace Validation files must be unique or they will not load into the Falcon software. It does not matter if they are in different folders.

From scratch using the provided template

Create a new test by selecting File→New→File and choosing a TraceValidator file and a basic template will be opened to start from. The template includes an Idle, Success, and Failed State along with simple Total, Success, and Fail counters. The Success and Fail States will increment the Total counter along with the Success or Fail counter. These are the minimum recommended objects that should be included in every TV file. With this template, you will add states and event details that get you from Idle to Success or Fail.

Custom Trace Validation tests must be saved in "C:\Users\[user name]\Documents\Protocol Insight\Falcon\CustomTests" in a subfolder created by the user. For example, a TV test named MyTraceValidation that is saved to a subfolder called "C:\Users\[user name] \Documents \Protocol Insight\Falcon\CustomTests\MyTests" will be displayed in the Test list in the Falcon software as

MyTraceValidation under the MyTests category.

***Important**: The name of Trace Validation files must be unique or they will not load into the Falcon software. It does not matter if they are in different folders.*

Trace Validation File Types

Each Trace Validation test case is made up of three files serving different functions. The XML file used by the Falcon software will be saved as a .piv (Protocol Insight Validation) file, and.sm and .sm.diagram files will also be saved. Each of the files makes use of XML and is therefore readable and editable in any XML or Text editor. Due to their complexity, it is highly recommended that files are only modified using Trace Validation Builder.

State Machine File (.sm)

The State Machine file has a .sm extension. These are the files that users open to directly edit with Trace Validation Builder. They contain all the pertinent data related to the State Machine and its objects.

Trace Validation XML File (.piv)

The Trace Validation XML output file is generated by Trace Validation Builder when you save a State Machine File with no errors, and is saved as a *.piv file. This file contains all the information that the Falcon software requires in order to validate trace data using a Trace Validation State Machine. These files are generated each time a user saves and will only be generated if the State Machine contains no errors.

State Machine Diagram File (.sm.diagram)

The State Machine Diagram file has a .sm.diagram extension. These files are not directly edited, but they are required. They will exist in the same directory with the same name as the State Machine file. They contain all the location and size data necessary to display objects visually on the diagram in Trace Validation Builder.

Basic Trace Validation Example

The following example will further demonstrate the concepts covered in this help manual. This takes the simple case of a device sending a data frame and the receiving device sending an AFC. It assumes that there have been no other packets sent and that the receiving device will send an AFC for every data frame.

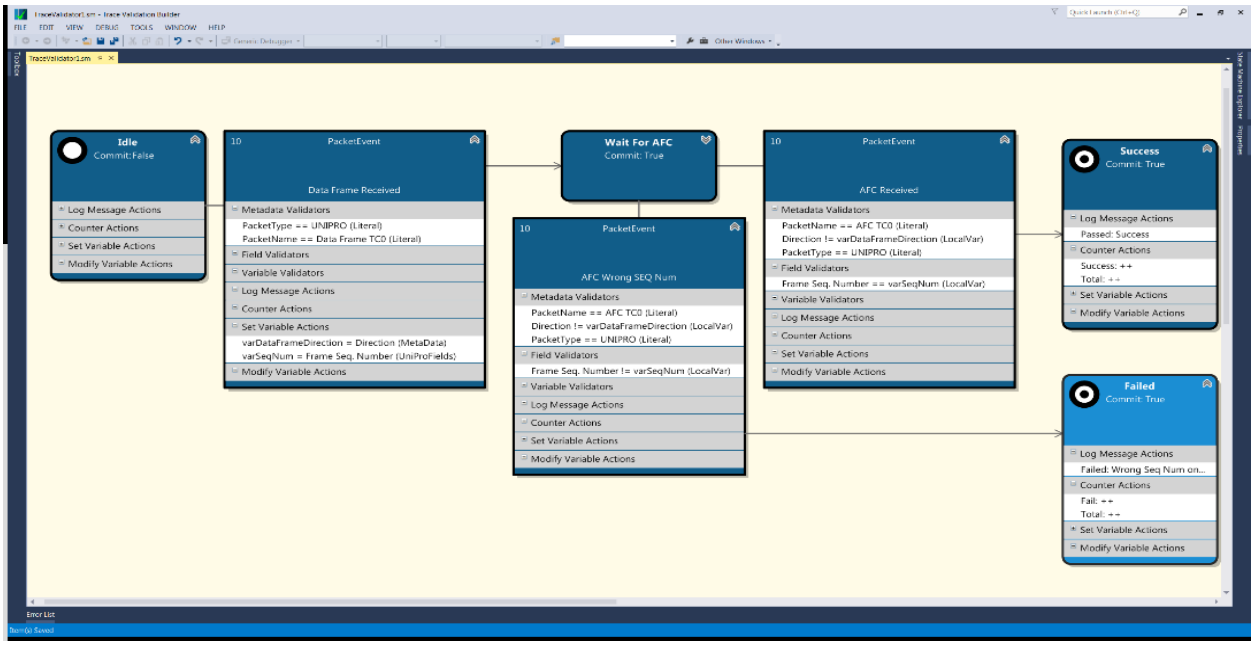

Figure 1: State Machine Example

1. **Initial State**
   When this State Machine is run, it will start in the *Idle* state and wait for its Event to fire. You will also see we have 3 counters (*Total, Success, and Fail*) as well as 2 variables (*varDataFrameDirection, varSeqNum*) defined.
2. **Data Frame Received Event**
   Using Metadata Validators, the *Data Frame Received* event is set to fire when it sees a UniPro Data Frame TC0 packet. It will then set both variables putting the direction the frame was seen into *varDataFrameDirection* and the Frame Seq.Number into *varSeqNum*. The State Machine then transitions to the *Wait for AFC* state.
3. **Wait For AFC State**
   This is a commit state so any counters or log messages that have be set up to this point will be saved. The State Machine now waits for one of the *Wait for AFC* State's Events to fire.
4. **Wait For AFC Events**
   a. *AFC Received*
      Using validators, this event will fire if it sees a UniPro AFC TC0 packet in the opposition direction as the original Data Frame. Direction is checked by looking for a direction no equal to our direction variable. Using the *varSeqNum* variable, it also checks to see that the Frame Seq. Number matches that of the data frame. The State Machine then transitions to the *Success* state.
   b. *AFC Wrong Seq Num*

This event is almost identical to the *AFC Received* event, the only difference is that is looks for Frame Seq. Number that does not match that of the data frame. If this event fires, we have an error. The State Machine then transitions to the *Failed* state.

5. **Final States**
   Depending on which Event fired in step 4, the State Machine will transition to one of the Final States, *Success* or *Failed*. In either case, a message is logged and the counters are incremented to indicate the success or failure.

Trace Validation Concepts

TV tests are built on a State Machine model. A state machine is a system that stores the status of something at a given time and, based on an input, can change the status. State machines can be described as discreet states with a set of possible events that will cause the system to move to a new state. States can also contain a set of actions or output events that trigger based on changing input.

Constructing UniPro vs. UFS TVs

Each TV must be defined as either a UniPro or UFS TV in Trace Validation Properties. Depending on the protocol chosen the user will be presented with different Event Validators for constructing the TV.

To access Trace Validation Properties right-click in the Diagram, the yellow background area of Trace Validation Builder, and select Properties. Select the appropriate "Protocol" from the Properties window. See THE DIAGRAM and THE PROPERTIES WINDOW for more information.

State Machine

The highest level of a file is the State Machine. There is only one State Machine in each file and it represents the overall system through which data is passed for validation.

State

A State is a discreet status indicting where in the State Machine the validation process is at present. As events occur, the system will move from State to State and actions can trigger that update information in the system. All State Machines must begin with an Initial State in which the State Machine will wait for an Event to trigger in order to move to another State. State Machines can only be in one State at any given time.

Event

An Event defines a set of conditions (implemented as Validators) that, when all are true, will trigger a transition to another State. States can have more than one event but only one Event can ever fire for a given State. See the *Event Evaluation Priority* section for more detail.

Shared Event

Shared Events function in the same way as Events, except they can be tied to more than 1 state. This allows you to create a single common Event for something and many States can use it. For example, you might want to consider your test a failure if you reach the end of the trace file before the State Machine reaches Success or Failure. Using a Shared Event, you can set up validators to check for the End of Trace packet and link to that event from all the States in the State Machine. This keeps the diagram less cluttered and provides a single object that can be change rather than having multiple copied and pasted Events

hanging off all the States.

Validator

There are several kinds of Validators available in Events. These validators are used to look at packets or variables to validate the data. When all the Validators on an Event are true, it will cause the Event to fire.

Transition

Transitions define which State the State Machine should enter after an Event has occurred. Each Event can only have one associated Transition. More than one Transition may feed into a single State.

Action

Actions can occur when an Event fires or when a State is entered. Trace Validation Builder provides actions to Log Messages, Update Counters, and Set or Modify Variables.

Variable

Variables, both Local and Global, can be used to store information for reference in many areas of a State Machine.

Counter

Counters are used for statistics and allow the user to provide detail on how many times something happens in a State Machine. For example, how many times a State Machine ran, succeeded, or failed or how many of a particular type of packet were found in a trace.

Working with Trace Validation Builder

When users create a new TV test, they are provided with a very basic State Machine template. This template provides the basic building blocks to support either UniPro or UFS protocol, with an Initial State named Idle and two final States, Success and Failed. In addition, the template starts with Fail, Success, and Total Counters. Starting with this template, users select UniPro or UFS protocol and define States and Events to move from Idle to Success or Fail.

Rules on Values

Throughout Trace Validation Builder, users will be able to set values. Values exist for variables and validators and follow a common set of rules unless otherwise specified. Values can contain Numeric data in the form of Hex or Integer values or string values. To specify a Hex value, you must begin the value with “0x” followed be a string of valid Hex characters (0-9 and A-F). Values starting with 0x and containing invalid Hex characters will cause an error. You may also store integer data in the form of whole integer values. All other values will be treated as string data. Decimal data is not supported, so a value of 12.33 would be treated as a string.

Value Types

In most places where you can specify a Value, you will be required to choose a value type. The Value Type tells the engine what data you want to compare and controls how the data in the Value field will be interoperated. In addition, to minimize errors and assist in State Machine construction, Trace Validation Builder uses a list engine to provide the appropriate Values in most Value fields based on the Value Type selected. Types will be displayed in various ways and will be discussed in the appropriate sections later.

The Compare Type values you will see are:

- **Literal –** A value type of literal will cause the comparison to use the hard-coded value entered in the Value field.
- **MetaData –** This will cause the engine to compare some value to the specified MetaData field of the packet being examined.
- **Fields –** This will cause the engine to compare some value to the data in the specified field of the packet being examined.
- **LocalVar –** This will cause the engine to compare some value to value of the specified Local Variable.
- **GlobalVa**r **–** This will cause the engine to compare some value to value of the specified Global Variable.
- **Expression –** Some objects, such as variables, allow you to set their value using expressions. See the *Expressions* section later for more detail.

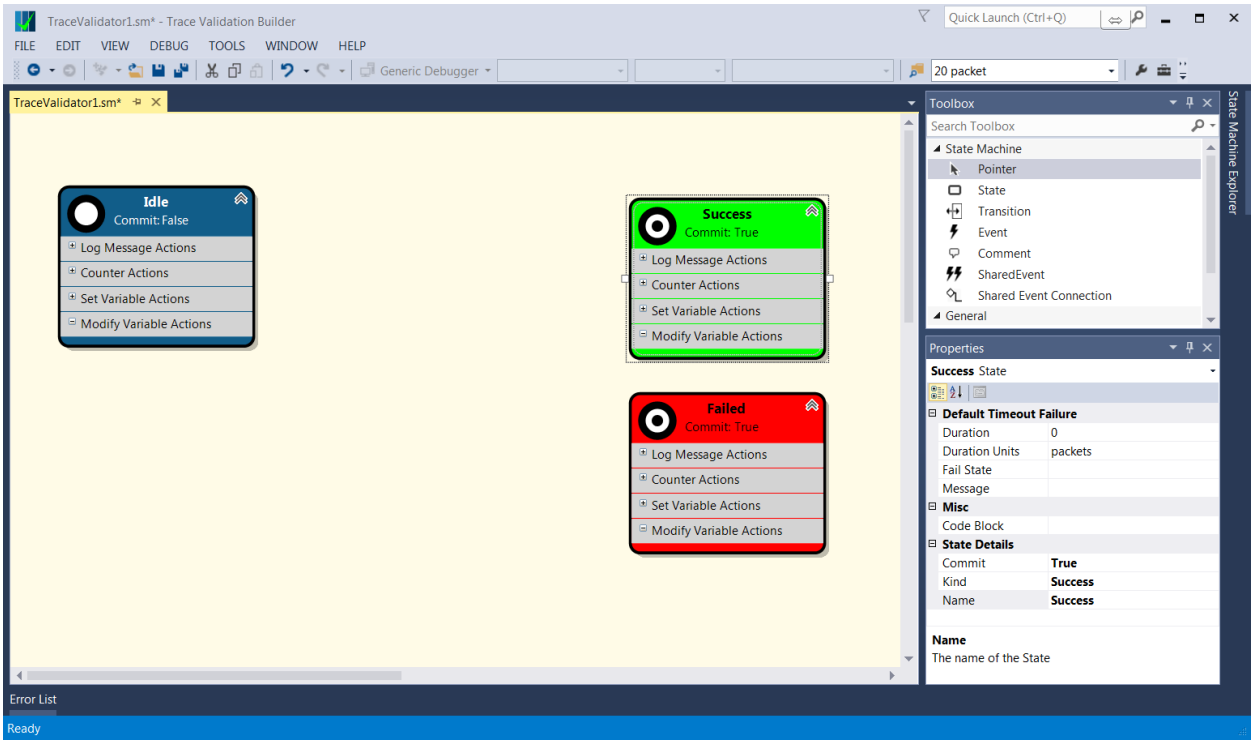

Figure 2: New TV test

The Diagram

The Diagram is the visual representation of a State Machine. It is the creative space in the file and is represented by the yellow background as seen in Figure 2. Each file contains information about one State Machine and therefore only has one Diagram.

The Toolbox

The Toolbox, which is on the right side of the application by default, contains most of the objects that you can add to a State Machine. To add a State, Shared Event, or Comment just click and drag that item from the toolbox to the Diagram. Events can only exist as children of States, so to add an Event, click and drag the Event toolbox item onto an existing State on the Diagram. Transitions define the transition between Events and States. To add a Transition, first select Transition in the toolbox, click the event you want the transition to come from, and finally click the State the want the Transition to go to. Shared Event Connectors are used to connect States to Shared Events and they work in a similar manor as transitions. Variables and Counters objects are also available in your State Machine Diagram but they are managed differently. See the Variables and Counters section for more detail.

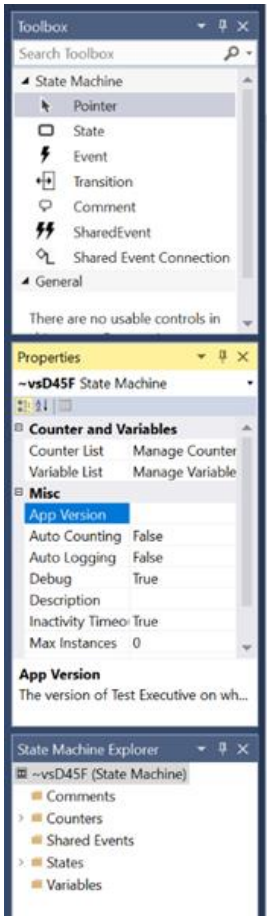

Figure 3: The Toolbox

The Properties Window

The Properties Windows is where the vast majority of the editing takes place in a State Machine. Therefore it is important that you leave this window open and available. If you should close it, you can open it again from the View→ Properties menu option or by pressing F4. The information displayed in the Properties Window will correspond to the item you have selected in the diagram.

The State Machine Explorer

The State Machine Explorer provides a simple way to find all the objects from your State Machine in a tree view.

Error List

The Error List will display a list of Warnings and Errors present in the State Machine. Warnings provide information on things that could cause unexpected behavior but will not prevent the Trace Validation from running in the Falcon software. Errors are items that need to be corrected as they will cause problems in the Falcon software. The Trace Validation XML file will not be generated if you have errors but you will be still be able to save your State Machine file.

Note: Warnings that start with "Cannot find a schema that defines target namespace" are internal and can be safely ignored.

State Machines

State Machines are the highest level parent in the file and contain all other objects. In order to select the State Machine, click anywhere on the yellow background of the diagram.

State Machine Properties

- **App Version** – A String value containing the application version for which this state machine was written. (Default: None)
- **Auto Counting** – A Boolean value which tells the State Machine whether or not to automatically add counters. When true, the engine will automatically add counters for Success, Fail, and Total. When false, the user must add any counters in the State Machine by using Counters and Counter

Actions. (Default: False)

- **Auto Logging**– A Boolean value which tells the State Machine whether or not to automatically log messages. When true, the engine will automatically log success and failure messages. When false, the user must log all messages in the State Machine by using Log Message Actions. (Default: False)
- **Counter List** – Clicking the ellipsis in this field opens the Counter Manager dialog. See the *Counters* section later for more detail.
- **Debug** – A Boolean value which tells the State Machine whether or not it can be run in debug mode. (Default: True)
- **Description** – A string value containing a description of the State Machine. (Default: None)
- **Forward** – A Boolean value which tells the State Machine whether to process the trace file in a forward or reverse direction. (Default: True)
- **Inactivity Timeout** – A Boolean value which tells the State Machine whether to timeout when the Global Inactivity Timeout is reached. (Default: True)
- **Max Instances** – Controls the number of instances that may be run at once. If MaxInstances is 0 then an unlimited number may run.
- **Protocol** – A string value containing the protocol for which this was written. (Default: UniPro)
- **Stimulus Test Case** – A string value containing the name of the corresponding Stimulus Test Case Category and Name that will generate an appropriate trace for this State Machine to validate. For Example: "MyCustomCases\MyCustomStimulusFile" will establish a link to the MyCustomStimulusFile test case in the MyCustomCases category. This will be the Stimulus case ran automatically before the TV analysis test in CTS mode. (Default: None)
- **Variable List** – Clicking the ellipsis in this field opens the Variable Manager dialog. See the *Variables* section later for more detail.

States

States are the primary building blocks of a State Machine. States define the current status of the State Machine and their Events define the next State to which the State Machine will move. When entering into a State, any actions on the State will run and then the State Machine simply waits for an Event to fire. There are four kinds of States available in a State Machine.

1. **Initial** – This is the beginning State. Each State machine may only have one Initial State. In general, no events will transition back into an Initial State.
2. **Normal** – The bulk of the States in your State Machine will be Normal. These States are transitioned to after an Event and they have their own Events which transition to other states.
3. **Success** – This is a Final State which should be reached if the validation of a trace was successful. Success Log Messages and Counters should be updated here. Success States cannot have Events.
4. **Fail** – This is a Final State which should be reached if the validation of a trace fails. Failure Log Messages and Counters should be updated here. Fail States cannot have Events.

State Properties

State Details

- **Name** – A String value containing the name of the State. This is for identification and each State must have a unique name. (Default: StateX)

- **Code Block** – Internal Use Only
- **Commit** – A Boolean value which tells the State Machine whether to commit Log messages and counters when this State is reached. Logs and counters will not show up in the application unless a State with Commit set to true has been reached. (Default: False)
- **Kind** – A String value which identifies the type of State as describe above. (Default: Normal)

Default Timeout Failure

- **Duration** – An Integer value specifying how long to wait in Duration Units for another Event to fire before timing out and moving to the specified Fail State. A value of 0 indicates that a default timeout is disabled. All other Default Timeout Failure values will be ignored when Duration is 0. (Default: 0)
- **Duration Units** – A string value specifying the type of units to wait for before timing out. You can wait for some number of packets or for an amount of time (nanoseconds, milliseconds, or microseconds). (Default: packets)
- **Fail State** – The State to move to when the timeout occurs. (Default: None)
- **Message** – The message to Log before moving to the specified failure state. (Default: None)

State Actions

You can define actions that will be executed when the State is entered during validation. There are four type of actions available, Log Message, Counter, Set Variable, and Modify Variable. To add an action, right click the compartment on the State for the type of Action you need and select Add new….. Alternatively, you can Right Click the State and select Add and then the type of item you want to add. In either case the new Action will appear in the appropriate compartment. You can select the new action and edit its properties in the Properties Window. Actions are deleted via Right Click and Delete or by selecting them and pressing the Delete key. The Actions available are described below.

Log Message Actions

Log a message to the Falcon software with the given Status and Text. The text is displayed in the Trace Validation Results window in the Falcon software along with an icon representing the status. For more information, see the *Interpreting Trace Validation Results* section in the Falcon Series User Manual.

The possible Status values are:

- **Failed**
  Used to indicate a failure in the Trace Validation. Messages of this type are usually logged just before transitioning to a Fail State or in a Fail State.
- **Warning**
  Warning messages are logged to indicate an issue that needs attention but that does not cause the validation to fail.
- **Passed**
  The Passed status is used to log messages after a successful validation has occurred. These types of messages are most often logged in a Success State.
- **Info**
  Info messages are used to provide further information to the Falcon software. These message are

not meant to indicate an issue with validation but simply to provide additional details or an explanation as to why something may not have been tested.

Counter Actions

Counter actions are used to update the value of a counter. In order to use a Counter Action, a Counter must be present in the State Machine, see the *Counters* section. When defining a Counter action, chose the counter from the list of available counters and then set a value. The Value property can be set to an integer value or to **++**. Setting a counter value to ++ tells the State Machine to add 1 to the current value of the counter.

Set Variable Actions

Set Variable Actions let you set or change a Variable's Value. In order to use a Set Variable Action, a Variable must be present in the State Machine, see the *Variables* section. Variables follow the rules outlined in the *Rules on Values* section above. To set a variable, select the Set Variable Action and in the Properties Window:

1. Choose the Variable to set
2. Select the Variable Value Type (See the *Value Type* section above)
3. Enter a Variable Value

When you choose a Variable Value Type of Expression, you are able to enter an expression into the Variable Value field. The resultant value will be stored in the variable. See the *Expressions* section for detail.

Modify Variable Actions

The Modify Variable Action is reserved for adding or subtracting from an Integer or Hex valued variable. Selecting an Action of ++ will increase the variable by 1, while selecting -- will decrease it by one. There is also a Max Value property which will allow the variable to wrap. For example, if the Max Value is 0xFF and the variable value is 0xFF, a ++ action would cause the variable to exceed the Max Value so it wraps back to 0. This is helpful when checking counters that wrap, such as UniPro Frame Sequence Number. This functionality also works for -- if the variable value is 0, it will wrap back to the Max Value. You can also set a variable value using the result of an Expression.

Events

Events are the main driving force behind a State Machine. You use Events to identify the data you are looking for in a trace, and they then allow you to move to the next State in the State Machine. There are two types of Events you can create, Packet Events which let you examine the packets in the trace, or Timeout Events which let you transition if a timer is exceeded before another Event fires.

Event Actions

There are actions available in Events to log messages, and work with counters and variables. The actions available within an Event are identical to those used in States, see the *State Actions* section above for more detail.

Event Properties

- **Name** – A String value containing the name of the Event. This is for identification and each Event

must have a unique name within its parent State. (Default: EventX)

- **Type** – A String value which indicates if this is a Packet Event or a Timeout Event. (Default: PacketEvent)
- **Next State** – A value which identifies the Name of the next State that this Event will transition to when firing. *Note: In general Next State is not set via this property, this will be set when creating a Transition.* (Default: None)
- **Evaluation Priority** – An Integer value that identifies the order in which the Events are to be evaluated, see *Event Evaluation Priority*. (Default: 10)
- **Duration** – *Only used for Timeout Events.* An Integer value specifying how long to wait in Units for another Event to fire before this timeout Event fires. (Default: 0)
- **Units** – *Only used for Timeout Events.* A string value specifying the type of units to wait for before timing out. You can wait for some number of packets or for an amount of time (nanoseconds, milliseconds, or microseconds). (Default: packets)

Validators

Validators are objects which allow the user to check data in a trace and validate it against some other source. When **all the Validators** in an Event are True, the Event will fire and transition to the next State. Data can be validated against other fields, metadata, variables, or user supplied values. There are three types of Validators that can be added to an Event, Metadata Validators, Field Validators, and Variable Validators. They all work in a similar fashion, the principle difference being the type of data you want to validate.

Metadata Validators

Metadata Validators allow you to compare the Metadata of a packet to other data. The properties of a Metadata Validator are broken into two sections. The first, Validation Source, contains the Metadata Tag. This is the piece of Metadata from the packet being evaluated that you want to compare. The second section, Validation Target, allows the user to specify to what they want the packet's Metadata to be compared. There are three properties that work together for the Validation Target. First, the Compare Type specifies what type of data to compare (see the *Value Types* section above). Next, the Compare Value is the actual Value that will be compared. Finally, the Comparison property allows the user to select the type of comparison to perform. All Validators support the following Comparisons:

- == (Equal)
- != (Not Equal)
- > (Greater Than)
- < (Less Than)
- >= (Greater Than or Equal To)
- <= (Less Than or Equal To)

For Example, the following values would look for a Data Frame TC0 packet.

- Metadata Tag: PacketName
- Compare Type: Literal
- Compare Value: Data Frame TC0
- Comparison: ==

Field Validators

Field Validators allow you to compare the data in a field of a packet to other data. The properties of a Field Validator are broken into the same two sections as the properties of a Metadata Validator. The only difference is that instead of a Metadata Tag, a Field Name is required for the Validation Source.

Variable Validators

Variable Validators allow you to compare the data in a Variable to other data. The properties of a Variable Validator are broken into the same two sections as the properties of a Metadata Validator. The only difference is that instead of a Metadata Tag, a Variable is required for the Validation Source.

Event Evaluation Priority

Event Evaluation Priority allows the user to control how Trace Validation will handle Events when there are more than one attached to a single State. Since only one Event can ever fire for a State, this is an issue if there are multiple Events with Validators that all evaluate to True at the same time. In this case, the event with the lowest Evaluation Priority will fire and the others will be ignored. If multiple Events are true and have the same Evaluation Priority, there is no guarantee as to which Event will fire. This is more clearly explained with the following examples.

Example 1: One Event with all Validators True

Note: Evaluation Priority doesn't affect Events in this case.

Event Name	All Validators True?	Evaluation Priority	Event Fires?
Event 1	Yes	13	Yes
Event 2	No	1	No
Event 3	No	2	No
Event 4	No	3	No

Example 2: Multiple Event with all Validators True

Note: Evaluation Priority does affect Events in this case.

Event Name	**All Validators True?**	**Evaluation Priority**	**Event Fires?**
Event 1	**Yes**	**13**	**No**
Event 2	**No**	**1**	**No**
Event 3	**Yes**	**2**	**Yes**
Event 4	**Yes**	**3**	**No**

Example 3: Multiple Event with all Validators True and Same Priority

Event Name	All Validators True?	Evaluation Priority	Event Fires?
Event 1	Yes	10	?
Event 2	No	10	No
Event 3	Yes	10	?
Event 4	Yes	10	?

These examples illustrator how Event Evaluation Priority affects which Event fires. If there is ever a case where multiple Events can have all their Validators evaluate to true (as is the case in Example 3), then specifying an Evaluation Priority is important to avoid unexpected results.

Transitions

Transitions simply tell the State Machine which State to move to after and Event fires. Transitions are represented in the Diagram as lines between Events and States. In order to tell the difference between a Transition and the connector between States and their child events, transitions have an arrowhead on the end that connects to a State.

Shared Events

Shared Events are configured almost identically to regular Events, with only a few differences. First, Shared Events can be linked to multiple States. Second, Evaluation Priority is not checked for Shared Events, and finally, they can be set to run before or after a States regular Events. To link a Shared Event to a State and control when they run, you use a Shared Event Connector.

Shared Event Connectors

Shared Event Connectors, much like Transitions, simply tell the State Machine that a State should use the connected Shared Event. Shared Event Connectors are represented on the graph as dashed lines between States and Shared Events. Shared Event Connectors also contain a diamond end on the State side of the connection with the Execution Group (either First or Last) listed next to the diamond. The Execution Group tells the State where in Evaluation Priority the Shared Events should be evaluated. Within the collection of Shared Events that a State references, there is no guaranteed order, so the best practice is to make sure that only one Shared Event can be evaluated to True at any given time. The Execution Group is modified via the Properties Window accessible when selecting the Shared Event Connector on the diagram. The follow lists the order in which Shared Events and Events will be evaluated for a given State:

1. All Shared Events in the First Execution Group – Order not guaranteed.
2. All regular Events owned by the State – Evaluated in Evaluation Priority Order.
3. All Shared Events in the Last Execution Group – Order not guaranteed.

Comments

Comments allow you to add a simple comment box to the diagram. This can be used as documentation or notes on the diagram to provide detail or explain a complicated state.

Variables

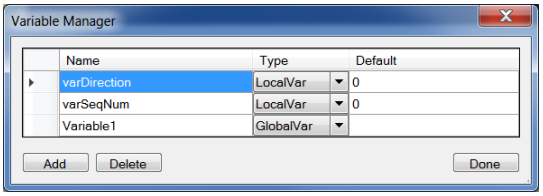

Figure 4: Variable Manager

Variables have been covered in several other areas as they are used to store data for later reference. You manage the variable with the Variable Manager dialog which you can access in two places. In the State Machine Properties there is a Property called Variable List with a Value or "Manage Variables". Selecting the ellipsis to the right of Manage Variables open the Variable Manager. Alternatively, you can right click a blank space on the diagram itself and select Manage Variables. Via this dialog, you can add a new variable by providing a name, type, and default value. Variables added here will be available in your State Machine. To delete a Variable, you must click the row selection box to the left of the Variable you wish delete and click the Delete button.

Variable Properties

- **Name** – A String value containing the name of the Variable. This is for identification and each Variable must have a unique name within State Machine. (Default: NewVariable)
- **Type** – A String value which indicates the type of variable. LocalVar indicates a local variable that can only be referenced in the current running instance of the State Machine in which it is defined. GlobalVar means a global variable that can be referenced from any running State Machine. (Default: LocalVar) ***Important: Because of the complexity of the interactions of Global Variables using them can lead to unexpected results. We recommend they be used only as a last resort and only by expert level users.***
- **Default** – This is the value to which the Local Variables will be set when the Initial State begins. Values can still be set in the Initial State by using a Set Variable Action to override the Default Value. Default Value follows the rules explained earlier in the *Rules on Values* section. Global Variables will always have a default value of 0 so you cannot set a default value in the Variable Manager. (Default: 0)

Counters

Counters have been covered in the *Counter Actions* section. Counters are managed like Variables, but via the Counter Manager. In the State Machine Properties click the ellipsis to the right of Manage Counters or right-click the diagram and select Manage Counters. Via this dialog, you can add a new counter by providing a name. Counters added here will be available in your State Machine. In order to delete a Counter, you must click the row selection box to the left of the Counter you wish delete and click the Delete button.

Expressions

Expressions are simple one-line code elements that can be used as validators or to set a variable's value. The result of any expression must be a single value. You can use expressions in two places, either to set the value of a variable in a Set Variable Action or in a Validator Expression. Code in a Set Variable Action can return any single value to be stored in the variable while code in Validator Expression must return a Boolean. Here are some examples of valid Validator Expressions which all evaluate to a Boolean.

- data.PacketMetaData["PacketName" == "PACP_SET_req"

- data.PacketFields["MIBattribute "] == "0x2041"
- data.PacketMetaData["Time"] - data.Locals["vTime"] > 5000

The following are examples of valid expression for Set Variable Actions which all return a single value.

- data.PacketMetaData["PacketType"]
- 2L + data.Locals["vCounter"]
- data.Locals["vCounter"] + data.Locals["vCounter"] – 10L

Values in Expressions

The values of the objects in your code are treated as either string, long, or Boolean. All string values are case sensitive. When comparing or setting long values, you will need to append "L" to any numeric value you enter. For example, data.Locals["vPacketCount"] = 50L.

Internal Dictionaries

There are a handful of dictionaries accessible via Expressions. They are all set up with the scope of the **data** object and all contain members which vary depending on the dictionary you choose.

data.PacketMetaData – This dictionary contains all the metadata for the packet. The members are the metadata fields that exist on each packet such as Direction, PacketType, PacketName, etc.

data.PacketFields – This dictionary contains all the field data for the packet. The members are all the fields that exist on each packet such as MIBAttribute, Frame Seq. Num, CReq, LUN, etc.

data.Locals - This dictionary contains all of the packages local variables. This dictionary can be used for both reading (get) and writing (set) variable values. The members are all the packages local variables.