Technical Reference

Tektronix Logic Analyzer Family TPI.NET Reference Manual Volume 1

071-1353-00

This document applies to TLA System Software Version 4.3 and above.

www.tektronix.com

TPI.NET Reference Manual

1 Introduction

TPI.NET

TPI.NET is a programmatic interface that exposes the system setup and acquisition data of a Tektronix Logic Analyzer. TPI.NET represents the TLA system through a hierarchy of software interface implementations. Clients of the programmatic interface must be .NET executables that run under the management of the .NET Common Language Runtime.

The TLA application supports two kinds of TPI.NET clients. Out-of-process client applications can use the .NET remoting infrastructure to connect to TPI.NET. The TLA also supports plug-in DLLs, which allow TPI.NET clients to execute as in-process extensions of the TLA application. See the document entitled **TPI.NET Remote Client Design Guide** for instructions on building TPI.NET client applications. See the document entitled **TLA Plug-In Design Guide** for information about creating plug-ins.

TPI.NET is available only on Version 4.3 and above of the TLA software. Client software that uses TPI.NET must be fully compatible with Version 1.0 of the Microsoft .NET Framework.

The software interfaces that are implemented by TPI.NET are defined in a .NET assembly named **TlaNetInterfaces.dll**. This file is part the TLA software installation and can be found in the following installation directory: C:\Program Files\TLA 700\System. TPI.NET client executables need to be compiled with a reference to this assembly. Since the assembly contains all the TPI.NET types, it is the only file that is needed to write client executables.

What This Document Contains

This document specifies the .NET types that are contained in the TlaNetInterfaces.dll assembly. To organize the large number of items specified, the document is divided into 11 sections:

1) Introduction.
2) TPI.NET Interfaces
3) Common Interfaces
4) Plug-In Interfaces
5) Structures (Value Types)
6) Exception Classes
7) Enumerations
8) Event Argument Classes
9) Delegates
10) Plug-In Support Classes
11) Appendices

Sections 2 through 4 contain .NET interface specifications, which are the bulk the document. The rest of the sections specify .NET constructs that are used to support the interface implementations.

This document assumes the reader is familiar with .NET programming. It does not attempt to provide tutorial information about the .NET Framework or writing managed .NET executables.

Samples

The same CD that contains this document also contains sample code for three example TPI.NET clients. The samples include a generic tool plug-in, a simple data window plug-in, and a remote TPI.NET client application. These samples are presented in three separate languages supported by Microsoft Visual Studio .NET: Visual Basic .NET, C#, and C++ with managed extensions.

Each sample has a solution file that can be opened and built to produce a working plug-in or client application.

The tool plug-in samples are contained in the following directories:
TPI.NET and PlugIn Documentation\Tool PlugIn Samples\CSharp Tool PlugIn,
TPI.NET and PlugIn Documentation\Tool PlugIn Samples\VB Tool PlugIn,
TPI.NET and PlugIn Documentation\Tool PlugIn Samples\CPP Tool PlugIn.

The data window plug-in sample is contained in the following directories:
TPI.NET and PlugIn Documentation\Data Window PlugIn Samples\CSharp Data Window PlugIn,
TPI.NET and PlugIn Documentation\Data Window PlugIn Samples\VB Data Window PlugIn.
TPI.NET and PlugIn Documentation\Data Window PlugIn Samples\CPP Data Window PlugIn,

The remote TPI.NET client application sample is in the following directories:
TPI.NET and PlugIn Documentation\Remote TPI.NET Client Samples\CSharp Remote TPI.NET Client,
TPI.NET and PlugIn Documentation\Remote TPI.NET Client Samples\VB Remote TPI.NET Client,
TPI.NET and PlugIn Documentation\Remote TPI.NET Client Samples\CPP Remote TPI.NET Client.

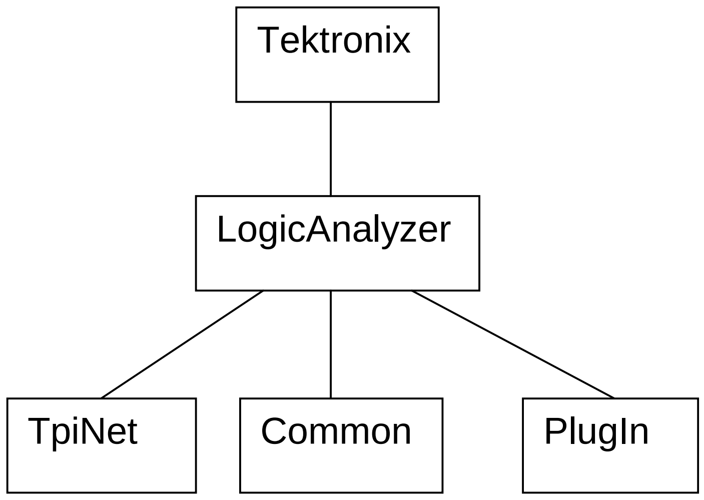
To use these sample projects, copy the directory contents of the desired sample from the CD into a directory on the computer that contains your development environment.

Please note that all the Visual Studio projects contain a pre-release version of the TlaNetInterfaces.dll assembly. If you experience any runtime problems related to TlaNetInterfaces.dll, you might get better results by compiling the samples, and any other TPI.NET clients, with the final release version of this assembly. The release version is always installed in the following directory of your TLA application software:

C:\Program Files\TLA 700\System\TlaNetInterfaces.dll

1.1 Namespaces

All types defined in the TlaNetInterfaces assembly are contained in one of three namespaces. TpiNet, Common, or PlugIn. Each of these namespaces is itself contained in the Tektronix.LogicAnalyzer namespace. The diagram below show the namespace structure of TlaNetInterfaces.dll.



For example the ITlaSystem interface (described in section 2.2) is contained within the TpiNet namespace. Its fully qualified named in the C# programming language is Tektronix.LogicAnalyzer.TpiNet.ITlaSystem.

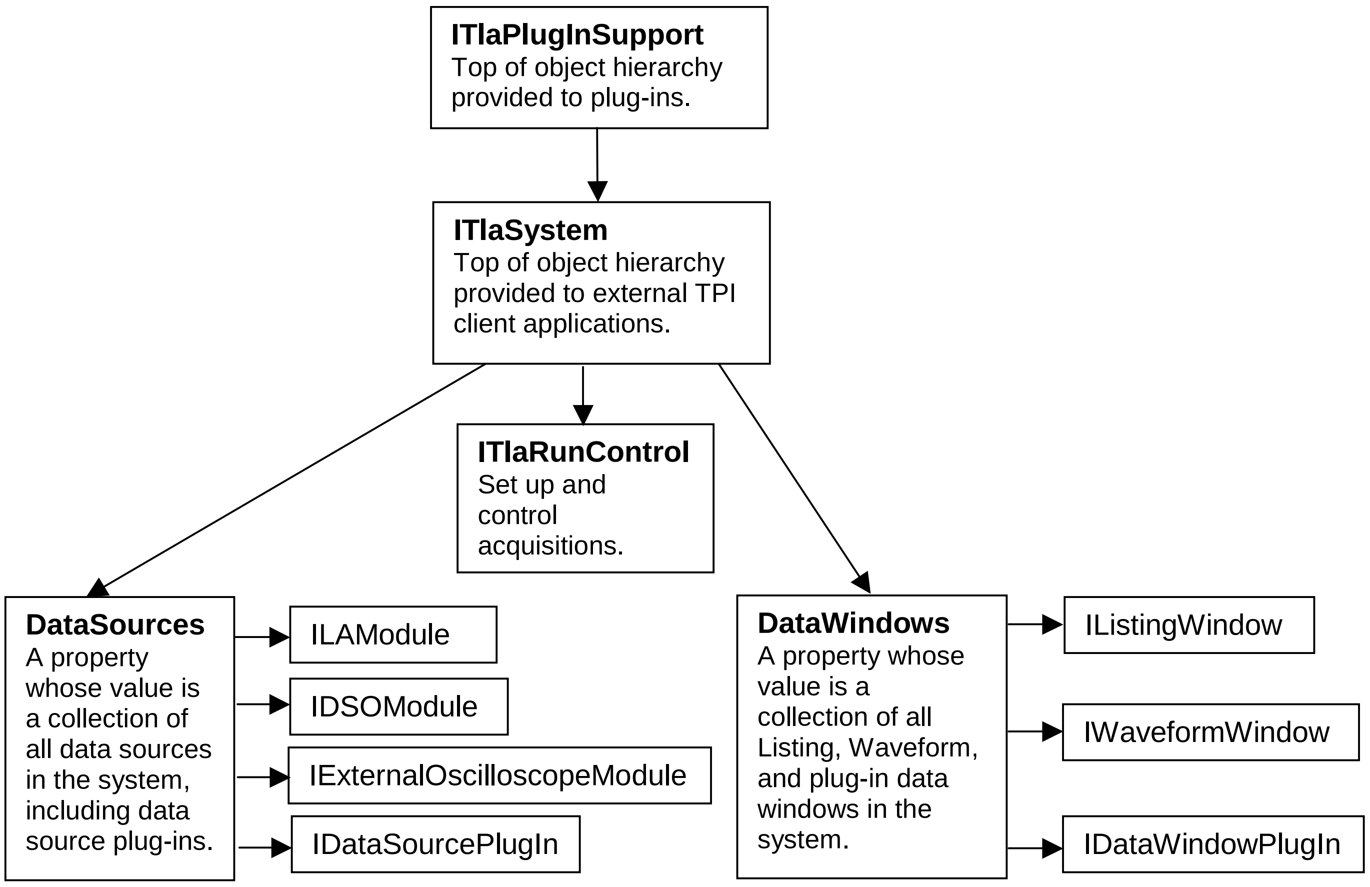
The **TpiNet** namespace contains the types that are primarily used to expose system setup and data. It has the interfaces implemented by the TLA application and the types used to support those interface implementations.

The **Common** namespace contains interfaces that are implemented by both the TLA application and by plug-in assemblies. Many types in the TpiNet namespace are derived from types in the Common namespace. In addition to interfaces, this namespace contains other types used by both the TLA application and by plug-ins.

The **PlugIn** namespace contains those Interfaces that must be implemented by plug-ins. It also contains optional plug-in interfaces and the Attribute classes that plug-ins can use to describe themselves.

1.2 TPI.NET Overview

TPI.NET is the term given to the hierarchy of .NET objects that provide programmatic access to the TLA. Both plug-ins and remote client applications can use this programmatic interface. However, plug-ins will need some services that are not needed by remote clients. For this reason there are two entry points into the TPI.NET object hierarchy. Access to an ITlaPlugInSupport object at the top of the hierarchy is provided only to plug-ins. Remote clients connect one level below through an ITlaSystem object. The figure below illustrates the top levels of the object hierarchy.



ITlaPlugInSupport
Top of object hierarchy provided to plug-ins.
ITlaSystem
Top of object hierarchy provided to external TPI client applications.
ITlaRunControl
Set up and control acquisitions.
DataSources
A property whose value is a collection of all data sources in the system, including data source plug-ins.
ILAModule
IDSOModule
IExternalOscilloscopeModule
IDataSourcePlugIn
DataWindows
A property whose value is a collection of all Listing, Waveform, and plug-in data windows in the system.
IListingWindow
IWaveformWindow
IDataWindowPlugIn

Both plug-ins and remote TPI clients programmatically access the TLA through an ITlaSystem object, which maintains collections of all the other objects that make up the system such as data sources and data windows.

The ITlaSystem object contains two important collections of objects: data sources and data windows. Those collections are discussed in the following two sections.

Data Sources

The property ITlaSystem.DataSources is a collection of all data sources in the system. Data sources that can be accessed through TPI are LA modules, DSO modules, an external oscilloscope, and data source plug-ins. All data sources accessible through TPI implement an interface that is derived from IDataSource.

TPI.NET represents LA and DSO modules as ILAModule and IDSOModule objects. An ILAModule allows for significant set up and control of a logic analyzer module. DSO modules are represented by IDSOModule and offer relatively less control over a DSO.

An external oscilloscope, if present, is exposed through an IExternalOscilloscopeModule object, which is identical to an IDSOModule object, except for the type name that is used to differentiate them.

Some data sources in the collection can be data source plug-ins. These are objects that implement IDataSourcePlugIn. When a data source plug-in is instantiated, it becomes part of the DataSources collection.

The DataSources collection is not the only way to retrieve references to data source objects. References to data sources can also be obtained by name; and LA and DSO modules can be obtained through their mainframe slot number.

Data Windows

The property ITlaSystem.DataWindows is a collection of all data windows in the system that are exposed through TPI.NET. Not all kinds of data windows are accessible through TPI.NET. For example, Histogram and Source windows are not.

Listing and Waveform windows are exposed as IListingWindow and IWaveformWindow objects. Control over these windows is primarily limited to moving cursors and locking windows together. The objects also expose the names of searches defined within the data window. Named searches can be activated programmatically, but searches cannot be created or edited.

Some data windows in the system can be data window plug-ins. These are objects that implement IDataWindowPlugIn. When a data window plug-in is instantiated, it becomes part of the DataWindows collection.

2 TPI.NET Interfaces

This section contains .NET interfaces that are implemented by the TLA application.

2.1 ITlaPlugInSupport

Description:

The TLA application implements this as the entry point for the support it provides to plug-in components. After the application creates a plug-in instance, it passes the new instance a reference to an ITlaPlugInSupport object in the IPlugIn.Initialize method. From an object of type ITlaPlugInSupport, a plug-in can access TPI.NET to set up and control the TLA; and it can obtain other plug-in support objects through TLA's implementation of IServiceProvider.

Namespace: Tektronix.LogicAnalyzer.TpiNet

Declaration Syntax:

Visual Basic

```
Interface IPlugInSupport
      Inherits IServiceProvider
```

C#

```
interface IPlugInSupport : IServiceProvider
```

C++

```
__gc __interface IPlugInSupport : public IServiceProvider
```

Remarks:

This interface gives access to portions of the TLA programmatic interface that are not available to remote TPI clients. Some plug-ins can get specific services by calling the GetService method of this interface.

2.1.1 ITlaPlugInSupport Properties

2.1.1.1 ITlaPlugInSupport.System

Description:

Gets a reference to an object of type ITlaSystem, which allows plug-ins to set up and control the TLA.

Declaration Syntax:

Visual Basic

```
ReadOnly Property System As ITlaSystem
```

C#

```
ITlaSystem System { get; }
```

C++

```
ITlaSystem* get_System ();
```

Arguments:

None.

Exceptions Thrown:

This property does not throw exceptions.

Remarks:

This property provides a reference to an object hierarchy representation of the TLA system. From this reference, plug-ins can programmatically control acquisitions, setup the system and its modules, and obtain data from data sources.

2.1.2 ITlaPlugInSupport Methods

2.1.2.1 ITlaPlugInSupport.GetService

Description:

Returns a service object that provides specialized support to plug-ins. This services provided by such objects go beyond the standard TPI.NET.

Declaration Syntax:

Visual Basic

```
Function GetService (serviceType as Type) As Object
```

C#

```
object GetService (Type serviceType)
```

C++

```
Object* GetService (Type* serviceType)
```

Arguments:

serviceType – The type of the service being requested.

Return Value:

Returns a reference to an object providing the requested service. If the requested type is not one of the services provided to plug-ins, then the reference is null. Otherwise the returned object is the type of the serviceType argument.

Exceptions Thrown:

This method does not throw exceptions.

Remarks:

This method is an implementation of IServiceProvider.GetService. The TLA uses this method to provide extensibility to it's plug-in support interface. Specialized support interfaces can be defined and implemented by the TLA application while leaving the ITlaPlugInSupport interface unchanged. Tektronix components, such as BusScope / BusDeskew, use this method to obtain the specialized services that they need to operate.

2.1.2.2 ITlaPlugInSupport.TopLevelFormToChild

Description:

This method makes a Form into a child of the TLA main application window. Plug-ins can use this method to integrate their windows into the application, making their forms into siblings of the TLA's own Setup and Trigger windows.

Declaration Syntax:

Visual Basic

`Sub` TopLevelFormToChild `(topLevelForm As Form)`

C#

`void` TopLevelFormToChild `(Form topLevelForm)`

C++

`void` TopLevelFormToChild `(Form* topLevelForm)`

Arguments:

`topLevelForm` – A reference to the Windows Form that is to be added to main application window. If `topLevelForm` is already a child of the main window, then the method has no effect.

Return Value:

None.

Exceptions Thrown:

None.

Remarks:

When a plug-in instantiates a Form object, it is by default a top-level window and not a child of the application's main MDI window. This method sets up integration of the form into the application MDI frame window. Because the main window is not itself a Form object, the TLA application manages certain aspects of the Form's behavior such as its maximized position, minimized position, bounds of movement, and z-order.

2.1.2.3 ITlaPlugInSupport.ChildFormToTopLevel

Description:

This method removes a child Form object from the main application window, and the Form becomes a top level window.

Declaration Syntax:

Visual Basic

```
Sub MakeFormTopLevel (childForm As Form)
```

C#

```
void MakeFormTopLevel (theForm childForm)
```

C++

```
Type-Name* Method-Name (theForm* childForm)
```

Arguments:

`childForm` – A reference to the Form object to turn into a top level window. If `childForm` is not currently a child of the main application window, then the method does nothing.

Return Value:

None.

Exceptions Thrown:

ArgumentNullException :

Condition: The `topLevelForm` argument is null.
Message: Null reference to Form object passed as argument.

Remarks:

The purpose of this method is to undo the effect of a previous call to ITlaUISupport.TopLevelFormToChild. After calling this method, the given Form will have no parent.

2.1.2.4 ITlaPlugInSupport.BlockRun

Description:

When a plug in calls this method, the TLA will not change from the stopped state until the calling object subsequently calls UnblockRun.

Declaration Syntax:

Visual Basic

```
Sub BlockRun (blocker As IPlugIn)
```

C#

```
void BlockRun (IPlugIn blocker)
```

C++

```
void BlockRun (IPlugIn* blocker)
```

Arguments:

`blocker` – A reference to the plug-in object that is requesting run blocking. This same reference must be passed to IProcessingSupport.UnblockRun to unblock acquisitions.

Return Value:

None.

Exceptions Thrown:

ArgumentNullException :
Condition: The `blocker` argument is null.

Remarks:

The application keeps a list of all clients that are blocking the next run, and the next acquisition will not start until that list is empty. Calls to this method by the same client are not cumulative. If a client mistakenly calls BlockRun more than once, then that client need only call UnblockRun once.

If the TLA is not in the stopped state when this method is called, the current acquisition will not be blocked. In that case, the call to BlockRun will take effect when the current acquisition completes, and the next acquisition will be blocked.

A call to BlockRun will affect iterations of a repetitive acquisition. This allows plug-ins to complete data processing between repetitive acquisitions.

2.1.2.5 ITlaPlugInSupport.UnblockRun

Description:

Plug-ins call this method to undo a previous call to BlockRun. UnblockRun removes a client from the list of clients that are currently blocking the next acquisition.

Declaration Syntax:

Visual Basic

```
Sub UnblockRun (blocker As IPlugIn)
```

C#

```
void UnblockRun (IPlugIn blocker)
```

C++

```
void UnblockRun (IPlugIn* blocker)
```

Arguments:

`blocker` – A reference to a plug-in that is currently blocking the next acquisition.

Return Value:

None.

Exceptions Thrown:

ArgumentNullException :
 Condition: The `blocker` argument is null.
 Message: Null reference to plug-in object passed as argument.

Remarks:

A plug-in should call UnblockRun only if it has previously called BlockRun. A call to UnblockRun with an invalid argument will have no effect.

2.1.2.6 ITlaPlugInSupport.ShowAddDataSourceDialog

Description:

This method displays the Add Data Source dialog, which allows users to add data source plug-in instances and data from saved modules to the system.

Declaration Syntax:

Visual Basic

```
Sub ShowAddDataSourceDialog ()
```

C#

```
void ShowAddDataSourceDialog ()
```

C++

```
void ShowAddDataSourceDialog ()
```

Arguments:

None.

Return Value:

None.

Exceptions Thrown:

TlaProhibitedDuringRunException :
Condition: This method is called during an acquisition.
Message: Cannot add data source during acquisition.

Remarks:

This method displays the Add Data Source dialog as if it had been selected from the application UI. Plug-ins can call this method to provide a consistent user interface for adding data sources to the system. Data window plug-ins in particular can call this method to help interactive users add data sources that will be displayed in the data window.

This method will not return until an interactive user has closed the Add Data Source dialog. To prevent the system from hanging while waiting for user input, plug-ins should only call this method in response to interaction with the plug-in UI.

2.1.2.7 ITlaPlugInSupport.ShowHelpTopic

Description:

This method opens a help file topic using the Win32 WinHelp utility. The file and the topic within the file are specified by arguments. Help topics within the TLA help manual can be displayed by passing a null or empty file argument.

Declaration Syntax:

Visual Basic

```
Sub ShowHelpTopic (topicID As Integer, file As String)
```

C#

```
void ShowHelpTopic (int topicID, string file)
```

C++

```
void ShowHelpTopic (int topicID, String* file)
```

Arguments:

topicID – The integer identifier of the help topic to show.

file – The full path to the help file that contains the desired help topic. If this argument is null or String.Empty, the TLA help file is used (tla700.hlp).

Return Value:

None.

Exceptions Thrown:

ApplicationException:
Condition: The call to WinHelp failed. This is very rare because WinHelp will succeed even if the arguments are bad.
Message: The WinHelp utility failed to open.

Remarks:

This method invokes the winhlp32.exe utility, which requires that a window be identified as the requester of the help topic. This method identifies the requester as the main application window.

To open an arbitrary help file using a secondary window, append the string ">secondary" where "secondary" is the name of the secondary window to use. If a null file is passed in, the TLA help file is opened using an overview style secondary window.

2.1.2.8 ITlaPlugInSupport.ShowOpenDialog

Description:

This method causes the application to display a TLA-style dialog box for opening files. The dialog allows users to select a file from the file system. The method returns the pointer to the string containing the full file path of the selected file and an indication whether the user cancelled out of the dialog box.

Declaration Syntax:

Visual Basic

```
Function ShowOpenDialog (title As String, filter As String _
      ByRef path As String) As Boolean
```

C#

```
bool ShowOpenDialog (string title, string filter, string ref path)
```

C++

```
bool ShowOpenDialog (String* title, String* filter, String** path)
```

Arguments:

`title` – The text to be shown in the title bar of the dialog. If this argument is null or empty, the generic title "Open" is used.

`filter` – This argument specifies filters that control the file types that are seen in the dialog. These are the selections that can be made in the "Files of Type" combo box of a file dialog. A filter string is specified by a series of sub-string pairs, each string being delimited by a '|' character. The first string specifies the text of the combo box item; the second string is the filter associated with the item. The end of the list is delimited by a "||" string.

For example, to specify filters for all files and for TLA setup files, the following filter argument could be used: "All files|*.*|TLA files|*.tla||"

If this argument is null or empty, then no items appear in the "Files of Type" combo box; and all files are shown.

`path` – This argument is passed by pointer. The value passed in is used as the path of initial directory displayed in the dialog box. The value passed out is the string pointer to the full path of the file selected at the time the dialog was closed.

If this argument is null or empty, then the default location for saving files is used. In version 4.3 of the TLA application, this location is "C:\My Documents".

Return Value:

A Boolean value is returned. A true value indicates the user chose to open the selected file. A false value means the user cancelled the dialog box without choosing to open the selected file. When the method returns, a pointer to a string containing the full path of the selected file is contained in the "path" argument.

Exceptions Thrown:

None.

Remarks:

The dialog displayed by this method is the same in size and appearance as other "Load" and "Open" dialogs in the TLA application. At the bottom of the dialog, there is a "Comment" box, which displays information about the saved systems or modules when a .TLA file is selected. Plug-ins can use this method to provide a dialog box that is consistent with the look and feel of the rest of the application.

2.1.2.9 ITlaPlugInSupport.GetSavedPlugInIDs

Description:

This methods allows a client to discover the string IDs of all plug-ins of a given type that are saved within a .tla file. The IDs can be used in subsequent calls to ITlaPlugInSupport.GetSavedData to obtain access to a saved plug-in with a file.

Declaration Syntax:

Visual Basic

```
Function GetSavedPlugInIDs (filePath As String, plugInType As Type) _
      As String()
```

C#

```
String[] GetSavedPlugInIDs (string filePath, Type plugInType)
```

C++

```
String* GetSavedPlugInIDs (String* filePath, Type* plugInType) []
```

Arguments:

`filePath` – The full path of the .tla system file that will be searched for saved plug-in IDs.

`plugInType` – The Type object for the kind of plug-ins for which the method should search.

Return Value:

A string array is returned. Each string in the array is a unique ID for a plug-in whose setup data was saved into the file. If the file contains no saved plug-ins of the given type, then an empty array is returned.

Exceptions Thrown:

TlaFileOperationException:
Condition: The exception is thrown when the specified file path does not refer to a valid saved TLA setup file.
Message: The specified file could not be opened.

Remarks:

The string IDs returned by this method all refer to saved instances of the specified type. The IDs do not give any information about the individual saved instances. Setup data from each saved instance can be accessed by obtaining a PlugInSavedInfo object from ITlaPlugInSupport.GetSavedData() and passing in the ID of a saved instance.

2.1.2.10 ITlaPlugInSupport.GetSavedData

Description:

This method is used to obtain a PlugInSavedInfo object, which provides access to the setup data of a saved plug-in in a .tla file. The PlugInSavedInfo object is described in the Other Classes section of this document.

This method has two overloads. The first specifies the saved plug-in using a string ID that was obtained by a call to GetSavedPlugInIDs. The second overload specifies the plug-in by type, and it returns a PlugInSavedInfo object for the first saved object of that type found in the specified .tla file.

The returned PlugInSavedInfo object implements IDisposable. When the client is done with the object it needs to call PlugInSavedInfo.Dispose() .

Declaration Syntax:

Visual Basic

```
Function GetSavedData (filePath As String, plugInID As String) _
      As PlugInSavedInfo

Function GetSavedData (filePath As String, plugInType As Type) _
      As PlugInSavedInfo
```

C#

```
PlugInSavedInfo GetSavedData(string filePath, string plugInID)

PlugInSavedInfo GetSavedData(string filePath, Type plugInType)
```

C++

```
PlugInSavedInfo* GetSavedData(String* filePath, String* plugInID)

PlugInSavedInfo* GetSavedData(String* filePath, Type* plugInType)
```

Arguments:

`filePath` – The full path of the .tla file that contains the saved plug-in setup data.

`plugInID` – The unique string ID of the saved plug-in for which setup data is desired. This value must be obtained by a previous call to GetSavedPlugInIDs.

`plugInType` – The type of plug-in for which a PlugInSavedInfo object is requested. If the file contains more than one saved instance of this type, access will be provided for the first instance found in the file.

Return Value:

A PlugInSavedInfo object is returned. This object has an 'Info' property of type SerializatonInfo. Use the 'Info' property to access the saved setup data of the plug-in instance. Calls on this

SerializationInfo object will deserialize saved values in the exact same manner as if they were called from the SerializationInfo object passed to a deserialization constructor.

If the file does not contain a plug-in that matches the ’plugInID’ or ’plugInType’ argument, then null is returned.

The returned PlugInSavedInfo object must be disposed when no longer needed.

Exceptions Thrown:

TlaFileOperationException:

Condition: The exception is thrown when the specified file path does not refer to a valid saved TLA setup file.

Message: The specified file could not be opened.

Remarks:

Neglecting to call Dispose on the returned PlugInSavedInfo object will keep the file handle in use indefinitely. To make sure that limited operating system resources are not wasted, call Dispose as soon as possible.

2.1.3 ITlaPlugInSupport Events

2.1.3.1 ITlaPlugInSupport.SystemIdle

Description:

Plug-ins that need to do background processing can subscribe to this event to receive notification when idle time is available from the TLA system.

Declaration Syntax:

Visual Basic

```
Event SystemIdle As EventHandler
```

C#

```
event EventHandler SystemIdle;
```

C++

```
__event EventHandler SystemIdle;
```

Event Data:

None.

Remarks:

Plug-ins that do background processing should subscribe to this event instead of Application.Idle. The TLA application is not a purely .NET application, and idle time is more reliably delivered from this event than from Application.Idle.

The TLA application cannot processes Windows messages while plug-ins are handling SystemIdle events. Only small chunks of processing should be done within an idle time handler; otherwise performance of the TLA application will be degraded. Large background processing tasks should be accomplished by doing only small parts of the task each time the SystemIdle event is raised.

When a plug-in no longer needs idle time for background processing, it should immediately unsubscribe from this event.

2.2 ITlaSystem

Description:

This interface represents the TLA system, providing programmatic access to the TLA, to its instruments, and to plug-in components.

Namespace: Tektronix.LogicAnalyzer.TpiNet

Declaration Syntax:

Visual Basic

```
Interface ITlaSystem
      Inherits IValidity
      Inherits IDefault
```

C#

```
interface ITlaSystem
      : IValidity, IDefault
```

C++

```
__gc __interface ITlaSystem
      : public IValidity, IDefault
```

Remarks:

The TLA system is represented by a hierarchy of objects. All objects in the hierarchy are directly or indirectly accessibly through an ITlaSystem object.

The specific behavior of IDefault members in the TLA implementation of ITlaSystem is described in the in ITlaSystem methods and events.

2.2.1 ITlaSystem Properties

2.2.1.1 ITlaSystem.RunControl

Description:

Gets the system's run control object, which sets up system level acquisition characteristics and has methods for starting and stopping acquisitions.

Declaration Syntax:

Visual Basic

```
ReadOnly Property RunControl As ITlaRunControl
```

C#

```
ITlaRunControl RunControl { get; }
```

C++

```
ITlaRunControl* get_RunControl ();
```

Arguments:

None.

Exceptions Thrown:

None.

2.2.1.2 ITlaSystem.DataWindows

Description:

Gets a list of references to all the Listing, Waveform, and plug-in data window objects in the system. If the system contains no data windows, then the value of this property is null.

Declaration Syntax:

Visual Basic

```
ReadOnly Property DataWindows As ArrayList
```

C#

```
ArrayList DataWindows { get; }
```

C++

```
ArrayList* get_DataWindows ();
```

Arguments:

None.

Exceptions Thrown:

NotSupportedException
Condition: Attempt is made to change the list.
Message: Collection is read-only.

Remarks:

The value of this property is a read-only ArrayList object. Although the getting the property value never throws an exception, any attempt to modify the list will throw NotSupportedException. This means that the number of available data windows in the system cannot be changed by operations on this ArrayList. Some data windows can be added or removed by the following ITlaSystem methods: LoadDataWindow, CreatePlugInInstance, and DisposePlugInIstance.

The array contains references to Object types. Clients of the array can use reflection to determine the exact type of each data window. Objects in this array can be of the following IDataWindow-derived types: ILockingWindow, ISearchableWindow, IListingWindow, IWaveformWindow, IDataWindowPlugIn.

2.2.1.3 ITlaSystem.DataSources

Description:

Gets a list of all data sources that exist in the current system. If the system has no data sources, the value of this property is null.

Declaration Syntax:

Visual Basic

```
ReadOnly Property DataSources As ArrayList
```

C#

```
ArrayList DataSources { get; }
```

C++

```
ArrayList* get_ DataSources ();
```

Arguments:

None.

Exceptions Thrown:

NotSupportedException
    Condition: Attempt is made to change the list.
    Message: Collection is read-only.

Remarks:

The value of this property is a read-only ArrayList object. Although the getting the property value never throws an exception, any attempt to modify the list will throw an exception. This means that the number of available data sources in the system cannot be changed by operations on the ArrayList. Some data sources can be added or removed by the following ITlaSystem methods: LoadDataSource, UnloadDataSource, CreatePlugInInstance, and DisposePlugInIstance.

The array contains references to Object types. Clients of the array can use reflection to determine the exact type of each data source. Objects in this array can be of the following IDataSource-derived types: ILAModule, IDSOModule, IOscilloscopeModule, IDataSourcePlugIn, IAcquisitionPlugIn.

2.2.1.4 ITlaSystem.PhysicalLAModules

Description:

The value of this property is an array of all physical LA modules in the system. This property contains an element for each physical LA module even when the system contains merged modules. If the system contains no physical LA modules, the value of this property is null.

Declaration Syntax:

Visual Basic

```
ReadOnly Property PhysicalLAModules As ArrayList
```

C#

```
ArrayList PhysicalLAModules { get; }
```

C++

```
ArrayList* get_PhysicalLAModules ();
```

Arguments:

None.

Exceptions Thrown:

This property does not throw exceptions.

Remarks:

If PhysicalLAModules is not null, the value of this property is an array of type IPhysicalLAModule. Objects of type IPhysicalLAModule are used to set up system inter-probing and analog feeds.

2.2.1.5 ITlaSystem.PlugIns

Description:

Gets an array containing references to all plug-ins that currently exist in the system. If the system has no plug-in instances, the value of this property is null.

Declaration Syntax:

Visual Basic

```
ReadOnly Property PlugIns As ArrayList
```

C#

```
ArrayList PlugIns { get; }
```

C++

```
ArrayList* PlugIns ();
```

Arguments:

None.

Exceptions Thrown:

NotSupportedException
> *Condition*: Attempt is made to change the list.
> *Message*: Collection is read-only.

Remarks:

The value of this property is a read-only ArrayList object. Although the getting the property value never throws an exception, any attempt to modify the list will throw an exception. To add or remove plug-in instances, make calls to the ITlaSystem methods CreatePlugInInstance or DisposePlugInInstance.

There can be overlap between the DataSources, DataWindows, and PlugIns properties. For example a data source plug-in will appear in the DataSources list and in the PlugIns list.

2.2.1.6 ITlaSystem.PlugInTypes

Description:

Gets an array of a Type objects representing the plug-in types currently recognized by the TLA application.

Declaration Syntax:

Visual Basic

```
ReadOnly Property PlugInTypes As Type()
```

C#

```
Type[] PlugInTypes { get;}
```

C++

```
Type* get_PlugInTypes () __gc[];
```

Arguments:

Argument – Description of argument.

Exceptions Thrown:

None.

Remarks:

The value of this property is an array that has one Type object for every distinct plug-in class that is recognized by the system.

If a client changes this array, it will not affect the system in any way. Plug-in assemblies cannot be programmatically installed or uninstalled through TPI.

2.2.1.7 ITlaSystem.IsAppVisible

Description:

Gets or sets a Boolean value that determines whether the TLA application UI is visible.

Declaration Syntax:

Visual Basic

```
Property IsAppVisible As Boolean
```

C#

```
bool IsAppVisible { set; get; }
```

C++

```
bool get_IsAppVisible ();
void set_IsAppVisible (bool value);
```

Arguments:

This property is not indexed.

Exceptions Thrown:

This property does not throw exceptions.

Remarks:

By setting this property clients can show and hide the TLA application UI. Because there might be multiple plug-ins or multiple TPI.NET clients, no client should behave as if it had exclusive control over the value of this property.

2.2.1.8 ITlaSystem.MessageBoxesEnabled

Description:

Gets or sets a Boolean value that determines whether the TLA application UI should attempt to use confirmation dialog boxes for various operations. A remote client can set this property to false to keep confirmation message boxes from appearing in the TLA application UI. Such a message box can interfere with remote operation of the TLA application by blocking the application until the dialog is closed by human intervention.

Confirmation dialog boxes will be hidden if any of the following are true: Display Confirmation Messages has been set to No in the System Options Preferences, a remote (COM) TPI client is currently connected, or the MessageBoxesEnabled property has been set to false.

Declaration Syntax:

Visual Basic

```
Property MessageBoxesEnabled As Boolean
```

C#

```
bool MessageBoxesEnabled { set; get; }
```

C++

```
bool get_MessageBoxesEnabled ();
void set_MessageBoxesEnabled (bool value);
```

Arguments:

This property is not indexed.

Exceptions Thrown:

This property does not throw exceptions.

Remarks:

After setting this property to one value, a different value may returned by the getter. The getter returns false if a remote (COM) TPI client is connected or if the setter was called with a value of false. The getter returns true if there is no remote (COM) TPI client connected and the setter has either not been used or has been called with a value of true.

The initial state of this property is true.

2.2.1.9 ITlaSystem.SWVersion

Description:

Gets the version of the TLA application. The value is a string with the form “Major.Minor.Build”, for example, “4.3.75”.

Declaration Syntax:

Visual Basic

```
ReadOnly Property SWVersion As String
```

C#

```
string SWVersion { get; }
```

C++

```
String* get_SWVersion ();
```

Arguments:

None.

Exceptions Thrown:

None.

2.2.1.10 ITlaSystem.FilePath

Description:

Gets the full path of the current system file. This is the file path that will be used to save the system if the user selects "Save System" from the file menu. If the current system setup has not been loaded by a "Load System…" operation, then the value of this property is String.Empty.

Declaration Syntax:

Visual Basic

```
ReadOnly Property FilePath As String
```

C#

```
string FilePath{ get; }
```

C++

```
String* get_FilePath();
```

Arguments:

None.

Exceptions Thrown:

None.

Remarks:

When a system is loaded with a "Load System" operation, the application maintains the value of this property until another system is loaded or the system is defaulted. However, if the system is being loaded at start-up, the application does not maintain this value after the load completes. At startup, a plug-in will be able to get the system file path only during its Initialize method. Afterward the value of FilePath will be an empty string.

In all cases where a plug-in is restored as part of a system load, it will have access to the system path when its Initialize method is called. A plug-in can determine if it is being restored by keeping track of which constructor was used to create it. A plug-in deserialization constructor is only invoked during system loads. If the default constructor was used, a system load in not in progress.

2.2.1.11 ITlaSystem.NumberOfMainframes

Description:

Gets the number of mainframes in the current system.

Declaration Syntax:

Visual Basic

```
ReadOnly Property NumberOfMainframes As Integer
```

C#

```
int NumberOfMainframes { get; }
```

C++

```
int get_NumberOfMainframes ();
```

Arguments:

None.

Exceptions Thrown:

None.

Remarks:

This value of this property is always greater than or equal to 1. The total number of expansion mainframes is NumberOfMainframes - 1.

2.2.1.12 ITlaSystem.ExternalSignalIn

Description:

Gets or sets the internal signal that is connected to External Signal In.

Declaration Syntax:

Visual Basic

```
Property ExternalSignalIn As SignalsIn
```

C#

```
SignalsIn ExternalSignalIn { set; get; }
```

C++

```
SignalsIn get_ExternalSignalIn ();
void set_ExternalSignalIn (SignalsIn);
```

Arguments:

None.

Exceptions Thrown:

None.

Remarks:

The value of this property is always a member of the SignalsIn enumeration.

2.2.1.13 ITlaSystem.ExternalSignalOut

Description:

Gets or sets the internal signal that is connected to External Signal Out.

Declaration Syntax:

Visual Basic

```
Property ExternalSignalOut As SignalsOut
```

C#

```
SignalsOut ExternalSignalOut { set; get; }
```

C++

```
SignalsOut get_ExternalSignalOut ();
void set_ExternalSignalOut (SignalsOut);
```

Arguments:

None.

Exceptions Thrown:

None.

Remarks:

The value of this property must is always a member of the SignalsOut definition.

2.2.1.14 ITlaSystem.ExternalSignalOutLowTrue

Description:

Gets or sets the logical polarity for External Signal Out. When the property is set to TRUE, the External Signal Out is low when asserted

Declaration Syntax:

Visual Basic

```
Property ExternalSignlalOutLowTrue As Boolean
```

C#

```
bool ExternalSignlalOutLowTrue { set; get; }
```

C++

```
bool get_Property-Name ();
void set_Property-Name (bool);
```

Arguments:

None.

Exceptions Thrown:

None.

2.2.1.15 ITlaSystem.ExternalTrigInEnabled

Description:

Gets or sets the enabled state of the External Trigger In connector.

Declaration Syntax:

Visual Basic

```
Property ExternalTrigInEnabled As Boolean
```

C#

```
Bool ExternalTrigInEnabled { set; get; }
```

C++

```
bool get_ ExternalTrigInEnabled ();
void set_ ExternalTrigInEnabled (bool);
```

Arguments:

None.

Exceptions Thrown:

InvalidOperationException:

Condition: An external scope has been added to the system, and the state of the External Trigger In connector cannot be changed because it will be used by the system to facilitate triggering between the scope and the TLA.

Message: The enabled state of the External Trigger In connector cannot be changed due to external oscilloscope in the system.

Remarks:

When an external oscilloscope is connected using iView, the enabled state of the External Trigger In connector is controlled by the system and cannot be changed.

2.2.1.16 ITlaSystem.ExternalTrigOutEnabled

Description:

Gets or sets the enabled state of the External Trigger Out connector.

Declaration Syntax:

Visual Basic

```
Property ExternalTrigOutEnabled As Boolean
```

C#

```
Bool ExternalTrigOutEnabled { set; get; }
```

C++

```
bool get_ ExternalTrigOutEnabled ();
void set_ ExternalTrigOutEnabled (bool value);
```

Arguments:

None.

Exceptions Thrown:

InvalidOperationException:
- *Condition*: An external scope has been added to the system, and the state of the External Trigger Out connector cannot be changed because it will be used by the system to facilitate triggering between the scope and the TLA.
- *Message*: The enabled state of the External Trigger Out connector cannot be changed due to external oscilloscope in the system.

Remarks:

When an external oscilloscope is connected using iView, the enabled state of the External Trigger Out connector is controlled by the system and cannot be changed.

2.2.2 ITlaSystem Methods

2.2.2.1 ITlaSystem.Default

Description:

Defaults the TLA. This method has the same affect as choosing Default System from the TLA application’s File menu.

Declaration Syntax:

Visual Basic

```
Sub Default ()
```

C#

```
void Default ()
```

C++

```
void Default ()
```

Arguments:

None.

Return Value:

None.

Exceptions Thrown:

This method does not throw exceptions.

Remarks:

Calling this method causes the TLA to default the system. All LA , DSO, and external oscilloscope instruments are removed, then re-instantiated in their default states. Persistent plug-ins are removed and disposed, then re-instantiated in their default states. Non-persistent plug-ins are removed from the system and disposed.

2.2.2.2 ITlaSystem.GetDataSourceByName

Description:

Returns a reference to the data source that has a given user name.

Declaration Syntax:

Visual Basic

```
Function GetDataSourceByName (userName As String) As IDataSource
```

C#

```
IDataSource GetDataSourceByName (string userName)
```

C++

```
IDataSource* GetDataSourceByName (String* userName)
```

Arguments:

`userName`– The name of the component as it appears in the TLA UI.

Return Value:

A reference to the data source that has the given name is return. If no data source has that name, then the return value is null.

Exceptions Thrown:

ArgumentNullException :
> *Condition*: The `userName` argument is a null reference.
> *Message*: Null reference passed to GetDataSourceByName.

ArgumentException :
> *Condition*: The `userName` argument does not represent the name of any data source in the system.
> *Message*: <userName> does not exist in the system.

Remarks:

This method return references to objects only if they are present in the TLA System window and therefore have a meaningful user name. The system might contain data sources not available from the system window. Access to all data sources can be obtained from the ITlaSystem.DataSources collection.

2.2.2.3 ITlaSystem.GetDataSourceByName

Description:

Returns a reference to the data source that has a given file name and user name.

Declaration Syntax:

Visual Basic

```
Function GetDataSourceByName (filename As String, userName As String)
As IDataSource
```

C#

```
IDataSource GetDataSourceByName (string filename, string userName)
```

C++

```
IDataSource* GetDataSourceByName (String* filename, String* userName)
```

Arguments:

`filename` – File name of the datasource to be searched for.
`UserName` – The name of the component as it appears in the TLA UI.

Return Value:

A reference to the data source that has the given filename and username is returned. If no data source has that name, then the return value is null. The filename needs to include the entire path name as well.

Exceptions Thrown:

ArgumentNullException :
Condition: The `filename` or `username` argument is a null reference.
Message: Null reference passed to GetDataSourceByName.

ArgumentException :
Condition: The `filename` and `username` argument does not represent the name of any data source in the system.
Message: unable to find datasource.

2.2.2.4 ITlaSystem.GetModuleBySlot

Description:

Returns a reference to the module that is located in a specific slot within the mainframe or one of the system's expansion mainframes.

Declaration Syntax:

Visual Basic

```
Function GetModuleBySlot (slot As Int32) As IDataSource

Function GetModuleBySlot (slot As Int32, mainframe As Int32) _
      As IDataSource
```

C#

```
IDataSource GetModuleBySlot (Int32 slot)

IDataSource GetModuleBySlot (Int32 slot, Int32 mainframe)
```

C++

```
IDataSource* GetModuleBySlot (Int32 slot)

IDataSource* GetModuleBySlot (Int32 slot, Int32 mainframe)
```

Arguments:

`slot` – A mainframe slot number.

`mainframe` – The number of the mainframe. The controller mainframe is number 0, the first expansion mainframe is number 1, and so on.

Return Value:

Returns a reference to an IDataSource object that represents a module or set of merged modules. The data source is either an LA or a DSO. Pattern generators cannot be accessed through TPI.

Exceptions Thrown:

ArgumentException :
Condition: Invalid slot number.
Message: The argument does not represent a valid physical slot in the specified mainframe.

Condition: Invalid mainframe number.
Message: The argument does not represent a valid mainframe in the TLA system.

Remarks:

This method is overloaded. If the one argument overload is used, then only the slot number is specified and the mainframe number is assumed to be 0. If the system has expansion mainframes, then the two argument overload can be used to specify the mainframe.

GetModuleBySlot can be used only to get data sources that are physically installed in the system. Use the ITlaSystem.DataSources collection to obtain references to all data sources in the system.

2.2.2.5 ITlaSystem.GetDiagnosticStatus

Description:

Returns a value that indicates the status of system diagnostics.

Declaration Syntax:

Visual Basic

```
Function GetDiagnosticStatus () As DiagnosticStatus
```

C#

```
DiagnosticStatus GetDiagnosticStatus ()
```

C++

```
DiagnosticStatus GetDiagnosticStatus ()
```

Arguments:

None.

Return Value:

An enumeration is returned to indicated the diagnostic status of the system. If the return value is not DiagnosticStatus.Pass, then the TLA instrument is not likely to perform properly.

Exceptions Thrown:

None.

Remarks:

This method can be used in combination with GetCalibrationStatus to determine if there are problems with the system that will prevent proper operation.

2.2.2.6 ITlaSystem.GetCalibrationStatus

Description:

Returns a value that indicates the calibration status of the system.

Declaration Syntax:

Visual Basic

```
Function GetCalibrationStatus () As CalibrationStatus
```

C#

```
CalibrationStatus GetCalibrationStatus ()
```

C++

```
CalibrationStatus GetCalibrationStatus ()
```

Arguments:

None.

Return Value:

A member of the enumeration CalibrationStatus is returned. If the return value is not CalibrationStatus.Calibrated, then the TLA instrument is not likely to perform properly.

Exceptions Thrown:

This method does not throw exceptions.

Remarks:

This method can be used in combination with GetDiagnosticStatus to determine if there are problems with the system that will prevent proper operation.

2.2.2.7 ITlaSystem.CreatePlugInInstance

Description:

Creates an instance of a specified plug-in and adds it to the current TLA system.

Declaration Syntax:

Visual Basic

```
Function CreatePlugInInstance (plugInType As Type) As IPlugIn
```

C#

```
IPlugIn CreatePlugInInstance (Type plugInType)
```

C++

```
IPlugIn* CreatePlugInInstance (Type* plugInType)
```

Arguments:

`plugInType` – A reference to the Type object associated with the desired plug-in. This must be a type of class, not an interface type.

Return Value:

Returns a reference to the newly created plug-in instance. If a new instance could not be created, then the return value is null.

Exceptions Thrown:

ArgumentException :
Condition: The `plugInType` argument does not represent a plug-in type.
Message: Passed in Type object is not a plug-in type.

ArgumentNullException :
Condition: The `plugInType` argument is null.
Message: Null argument passed to CreatePlugInInstance.

TlaPlugInException :
Condition: `plugInType` is a single-instance plug-in, and there is already an instance of it in the system.
Message: Single instance plug-in type <class-name> already exists.

Remarks:

This method is used to create an instance of a plug-in from an assembly that is installed in the "PlugIns" directory of the TLA software installation. The caller passes a Type object to indicate what kind of plug-in to instantiate. The property ITlaSystem.PlugInTypes can be used to get a collection of Type objects for all available plug-ins.

Note that some plug-ins do not support multiple instances of themselves. If the plug-in to be created is already instantiated and is decorated with the attribute [PlugInInstantiation(true, true)], then the TLA application will throw an exception.

2.2.2.8 ITlaSystem.DisposePlugInInstance

Description:

Removes a plug-in instance from the current system and disposes it.

Declaration Syntax:

Visual Basic

```
Sub DisposePlugInInstance (plugIn As IPlugIn)
```

C#

```
void DisposePlugInInstance (IPlugIn plugIn )
```

C++

```
void DisposePlugInInstance (IPlugIn* plugIn )
```

Arguments:

`plugIn` – Reference to the plug-in instance to be disposed.

Return Value:

None.

Exceptions Thrown:

ArgumentNullException :
Condition: The `plugIn` argument is null.
Message: Null argument passed to DisposePlugInInstance.

Remarks:

The specified component is first removed from all ITlaSystem collections and is no longer part of the system. Then its Dispose method is called.

2.2.2.9 ITlaSystem.SaveSystem

Description:

Save the current system setup to a file. Instrument data can optionally be saved in the file, and a user comment can be added.

Declaration Syntax:

Visual Basic

```
Sub SaveSystem (fileName As String, comment As String, _
                data As SaveDataOptions)
```

C#

```
void SaveSystem (string fileName, string comment, SaveDataOptions data)
```

C++

```
void SaveSystem (String* fileName, String* comment,
                 SaveDataOptions data)
```

Arguments:

`fileName` – The full path of the file to be saved.

`comment` – A user comment to be save along with the file. This comment can be seen in Load System dialog in the application UI. This value can be either a null reference or an empty string.

data – a member of the SaveData enumeration that determines whether system data is saved to the file.

Return Value:

None.

Exceptions Thrown:

TlaFileOperationException :
Condition: The system was running during the save operation.
Message: The file operation could not complete because the system was running.

ArgumentNullException :
Condition: The `fileName` argument is null.
Message: Null argument passed to SaveSystem.

ArgumentException :
Condition: The `fileName` argument is not a valid filename.
Message: Invalid file name passed to SaveSystem.

PathTooLongException :
Condition: The file path contained in `filename` is too long.

Message: The passed-in file name is too long.

UnauthorizedAccesException :
Condition: The specified file is read-only.
Message: The specified file is read-only.

Remarks:

Since many problems can occur during file operations, the SaveSystem method should always be called within a try block, so that exceptions can be caught. The above list of possible exceptions cannot be guaranteed complete, so a catchall clause can be used in addition to catching specific exceptions of interest.

2.2.2.10 ITlaSystem.LoadSystem

Description:

Loads a saved system setup from a specified file.

Declaration Syntax:

Visual Basic

```
Sub LoadSystem (fileName As String)
```

C#

```
void LoadSystem (string fileName)
```

C++

```
void LoadSystem (String* fileName)
```

Arguments:

`filename` – The full path to the saved system file to be loaded.

Return Value:

None.

Exceptions Thrown:

TlaFileOperationException :
Condition: Several file problems, not directly related to the file system, can cause the TLA to throw this exception. The exception can be examined to determine what problem was at fault.
Message: <The message depends on the failure reason.>

ArgumentNullException :
Condition: The `fileName` argument is null.
Message: Null argument passed to LoadSystem.

ArgumentException :
Condition: The `fileName` argument is not a valid filename.
Message: Invalid file name passed to LoadSystem.

PathTooLongException :
Condition: The file path contained in `filename` is too long.
Message: The passed-in file name is too long.

Remarks:

Since many problems can occur during file operations, the LoadSystem method should always be called within a try block so exceptions can be caught. The above list of possible exceptions cannot be guaranteed complete, so a catchall clause can be used in addition to catching specific exceptions of interest.

2.2.2.11 ITlaSystem.LoadSymbolFile

Description:

This method loads a specified symbol file into the system. When loading files with range symbols, the method is overloaded so that range symbols can be loaded with default options; or an object of type RangeSymbolOptions can be passed to specify how range symbols should be loaded.

The method creates a new ISymbolFile object and returns a reference it. The returned object allows basic translations of bit patterns into strings by using symbol tables from the file. When LoadSymbolFile is called and the `preventUnload` argument is true, the returned object will have its ISymbolFile.IsPreventingUnload property automatically set to true. This ensures that the symbol file will be available as long as the client needs it. Clients should then set ISymbolFile.IsPreventingUnload to false as soon the symbol file is no longer needed.

If the symbol file is already loaded, this call reloads it. Any changes to the file that have occurred since the last load will be reflected by a later call to LoadSymbolFile.

Declaration Syntax:

Visual Basic

```
Function LoadSymbolFile (fileName As String, _
      preventUnload As Boolean) As ISymbolFile

Function LoadSymbolFile (fileName As String, _
      options As RangeSymbolOptions, preventUnload As Boolean) _
      As ISymbolFile
```

C#

```
ISymbolFile LoadSymbolFile (string fileName, bool preventUnload)

ISymbolFile LoadSymbolFile (string fileName,
      RangeSymbolOptions options, bool preventUnload)
```

C++

```
ISymbolFile LoadSymbolFile (String* fileName, bool preventUnload)

ISymbolFile LoadSymbolFile (String* fileName,
      RangeSymbolOptions options, bool preventUnload)
```

Arguments:

`fileName` – The full path of the symbol file to load.

`options` – A RangeSymbolOptions object that defines which range symbols should be loaded and what offsets should be applied to the symbol values.

`preventUnload` – This Boolean parameter controls whether or not the returned ISymbolFile object prevents the target symbol file from being unloaded. When the value is true, the returned ISymbolFile instance has its IsPreventingUnload property set to true. Otherwise the property is false.

Return Value:

Returns an ISymbolFile object that can be used to translate bit patterns into symbolic strings.

Exceptions Thrown:

TlaFileOperationException :

Condition: The symbol file is not valid. When this is thrown the Failure member will be set to `InvalidFile`.

Message: The symbol file does not have a valid or recognizable format.

ArgumentNullException :

Condition: The `fileName` argument is null.

Message: Null argument passed to LoadSymbolFile.

ArgumentException :

Condition: The `fileName` argument is not a valid filename.

Message: Invalid file name passed to LoadSymbolFile.

PathTooLongException :

Condition: The file path contained in `filename` is too long.

Message: The passed-in file name is too long.

FileNotFoundException :

Condition: Specified symbol file couldn't be found.

Message: Symbol file not found.

FileLoadException :

Condition: File was found, but can't be loaded.

Message: Symbol file cannot be loaded.

Remarks:

Since many problems can occur during file operations, the LoadSymbolFile method should always be called within a try block so exceptions can be caught. The above list of possible exceptions cannot be guaranteed complete, so a catchall clause can be used in addition to catching specific exceptions of interest.

When this method is successful, it always creates a new ISymbolFile object. Each ISymbolFile object is individually capable of preventing its associated symbol file from being unloaded. A symbol file cannot be unloaded until all the ISymbolFile objects that target it have their IsPreventingUnload properties set to false. Therefore clients that load symbol tables need to make sure that ISymbolFile.IsPreventingUnload is set to false before the client releases its references to ISymbolFile objects.

2.2.2.12 ITlaSystem.GetSystemTriggerTime

Description:

Gets the time in picoseconds of the last system trigger. This time is the number of picoseconds elapsed from the start of the acquisition until the system trigger.

Declaration Syntax:

Visual Basic

```
Function GetSystemTriggerTime () As Int64
```

C#

```
Int64 GetSystemTriggerTime ()
```

C++

```
Int64 GetSystemTriggerTime ()
```

Arguments:

None.

Return Value:

This method returns a signed 64 bit value that represents the number of picoseconds elapsed from the start of the acquisition until the system trigger. The value returned is always positive.

Exceptions Thrown:

TlaNoDataException:

Condition: The TLA has no acquisition data, so it has no system trigger time.
Message: No system trigger information available due to lack of acquisition data.

2.2.2.13 ITlaSystem.LoadDataSource

Description:

Adds a reference data source to the system from data in a saved file. The data are then available to data windows and to plug-ins.

Declaration Syntax:

Visual Basic

```
Function LoadDataSource (fileName As String, _
      dataSourceName As String) As IDataSource
```

C#

```
IDataSource LoadDataSource (string fileName, string dataSourceName)
```

C++

```
IDataSource* LoadDataSource (String* fileName, String* dataSourceName)
```

Arguments:

`fileName` – The full path of the file that contains the saved data source.

`dataSourceName` – This is name of the module or the name of a data source plug-in that is saved in the file.

Return Value:

If no exception is thrown, then a reference is returned to the specified data source.

Exceptions Thrown:

TlaNoDataException :
Condition: The specified module does not have any data saved in the file.
Message: Module <dataSourceName> does not have any saved data.

TlaFileOperationException :
Condition: Several file problems, not directly related to the file system, can cause the TLA to throw this exception. The exception can be examined to determine what problem was at fault.
Message: <The message depends on the failure reason.>

ArgumentNullException :
Condition: The `filename` argument is null.
Message: Null argument passed to LoadDataSource.

ArgumentException :
Condition: The `filename` argument is not a valid filename.
Message: Invalid file name passed to LoadDataSource.

PathTooLongException :

Condition: The file path contained in `fileName` is too long.
Message: The passed-in file name is too long.

Remarks:

Since many problems can occur during file operations, the LoadDataSource method should always be called within a try block so exceptions can be caught. The above list of possible exceptions cannot be guaranteed complete, so a catchall clause can be used in addition to catching specific exceptions of interest.

When a data source is loaded from the from a saved file, its data and setup are static and cannot be changed. Such a data source is called a “reference data source.”

If the requested data source is already part of the system, attempting to load it again will not cause an exception. A reference will be returned to the existing instance of the reference data source.

2.2.2.14 ITlaSystem.UnloadDataSource

Description:

Disposes of a reference data source that was added to the system by LoadDataSource.

Declaration Syntax:

Visual Basic

```
Sub UnloadDataSource (dataSource As IDataSource)
```

C#

```
void UnloadDataSource (IDataSource dataSource)
```

C++

```
void UnloadDataSource (IDataSource* dataSource)
```

Arguments:

`dataSource` – A reference to the data source to be unloaded.

Return Value:

None.

Exceptions Thrown:

ArgumentException :
Condition: The `dataSource` argument references a data source that cannot be unloaded.
Message: Attempt made to unload an a non-reference data source.

ArgumentNullException :
Condition: The `dataSource` argument is null.
Message: Null argument passed to UnloadDataSource.

2.2.2.15 ITlaSystem.SetSystemTrigger

Description:

Sets the system triggering behavior to one of three pre-defined system trigger setups. The triggering behavior is changed by altering the trigger action of all modules in the system that currently have a trigger action.

Declaration Syntax:

Visual Basic

```
Sub SetSystemTrigger (triggerOption As SystemTrigger, _
                      dataSource As IDataSource)
```

C#

```
void SetSystemTrigger (SystemTrigger triggerOption,
                       IDataSource datasource)
```

C++

```
void SetSystemTrigger (SystemTrigger triggerOption,
                       IDataSource* dataSource)
```

Arguments:

`triggerOption` – a member from the SystemTrigger enumeration that selects how the system trigger should be generated.

`dataSource`– When the triggerOption argument is set to SystemTrigger.SingleModule, the dataSource argument is a reference to the single module that should trigger the system. If triggerOption is not SystemTrigger.SingleModule, then the dataSource argument is ignored.

Return Value:

None.

Exceptions Thrown:

ArgumentNullException :

Condition: The triggerOption argument is set to SystemTrigger.SingleModule and the dataSource argument is a null reference.

Message: Null dataSource reference passed to SetSystemTrigger.

ArgumentException :

Condition: The triggerOption argument is set to SystemTrigger.SingleModule, but the referenced dataSource has no triggering ability. The data source contains static reference data or is a plug-in that doesn't participate in system triggering.

Message: The target data source does not participate in system triggering.

TlaSystemTriggerException :

Condition: The current state of the system prevents the requested change to the system trigger. This exception can be thrown even when all the arguments are valid.
Message: <Message strings are taylored to the specific problem.>

Remarks:

This method always throws an exception when it fails. The method can fail even if all the arguments are valid. One cause of problems is that no data sources in the system have trigger actions, in which case SetSystemTrigger cannot cause set up the system to generate a system. When this kind of problem is detected, a TlaSystemTriggerException is thrown.

2.2.2.16 ITlaSystem.CreateCorrelatorObject

Description:

This method creates a new ICorrelator object.

Declaration Syntax:

Visual Basic

```
Function CreateCorrelatorObject () As ICorrelator
```

C#

```
ICorrelator CreateCorrelatorObject ()
```

C++

```
ICorrelator* CreateCorrelatorObject ()
```

Arguments:

None.

Return Value:

Returns a reference to a new ICorrelator object.

Exceptions Thrown:

None.

2.2.2.17 ITlaSystem.CreateSystemOptionAccess

Description:

This method creates a new ISystemOptionAccess object.

Declaration Syntax:

Visual Basic

```
Function CreateSystemOptionAccess () As ISystemOptionAccess
```

C#

```
ISystemOptionAccess CreateSystemOptionAccess ()
```

C++

```
ISystemOptionAccess* CreateSystemOptionAccess ()
```

Arguments:

None.

Return Value:

Returns a reference to a new `ISystemOptionAccess` object.

Exceptions Thrown:

None.

2.2.2.18 ITlaSystem.CanMerge

Description:

This method is used to determine if it's possible to merge a given set of LA Modules. The client is returned a simple Boolean argument to indicate.

Declaration Syntax:

Visual Basic

```
Function CanMerge (moduleArray As ILAModule()) As Boolean
```

C#

```
Boolean CanMerge (ILAModule moduleArray);
```

C++

```
Boolean GetPossibleMerges (ILAModule*__gc[] moduleArray)
```

Arguments:

`moduleArray` – An array of modules to be tested to see if they can merge.

Return Value:

The method returns Boolean indicating whether the `moduleArray` can be merged into the system.

2.2.2.19 ITlaSystem.Merge

Description:

Programmatically merges two or more logic analyzer modules into a single module.

Declaration Syntax:

Visual Basic

```
Function Merge (moduleArray As ILAModule()) As ILAModule
```

C#

```
ILAModule Merge (ILAModule moduleArray)
```

C++

```
ILAModule* Merge (ILAModule*__gc[] moduleArray)
```

Arguments:

`moduleArray` – An array of modules to be merged into a single module.

Return Value:

If the merge is successful, a reference to the new module is return. If the merge fails for any reason, an exception is thrown.

Exceptions Thrown:

ArgumentNullException :
Condition: The array reference is null.
Message: Null array reference passed to Merge.

Condition: One of the array elements is null.
Message: Null array element passed to Merge.

ArgumentException :
Condition: The elements in the array do not constitute a set of modules that can be merged.
Message: The module set cannot be legally merged.

ApplicationException :
Condition: The modules set is legal but there is an unexpected hardware problem.
Message: Cannot merge modules due to unexpected hardware problem.

Remarks:

Merges that would otherwise be legal can fail due to hardware problems. The application will throw an exception when this happens, and client should always be prepared to catch the exception.

2.2.2.20 ITlaSystem.Unmerge

Description:

Takes a merged instrument and unmerges it into a set of ILAModule objects, one for each physical module that comprised the merged module.

Declaration Syntax:

Visual Basic

```
Function Unmerge (module As ILAModule) As ILAModule()
```

C#

```
ILAModule[] Unmerge (ILAModule module );
```

C++

```
ILAModule* Unmerge (ILAModule* module ) __gc[];
```

Arguments:

`module` – The merged module that is to be unmerged.

Return Value:

The unmerged set of modules is returned as an array of ILAModule objects.

Exceptions Thrown:

ArgumentNullException :
 Condition: The module argument is null.
 Message: Null argument passed to Unmerge.

Remarks:

The return format is maintained even if an unmerged module is passed in, in which case the returned array contains only one module.

2.2.2.21 ITlaSystem.GetLockedWindows

Description:

This method gets an array of all windows currently locked to each other in the TLA application.

Declaration Syntax:

Visual Basic

```
Function GetLockedWindows () As ILockingWindow()
```

C#

```
ILockingWindow[] GetLockedWindows ();
```

C++

```
ILockingWindow* GetLockedWindows () __gc[];
```

Arguments:

None.

Return Value:

Returns an array of ILockingWindow objects unless no windows in the system are currently locked. If there are no locked windows, then a null reference is returned.

Exceptions Thrown:

None.

Remarks:

Changing the values of in the Locked array does not affect which windows in the system are locked. To change the set of locked data windows, call ITlaSystem.SetLockedWindows.

2.2.2.22 ITlaSystem.SetLockedWindows

Description:

This method takes an array of ILockingWindow objects and makes them the current set of locked windows in the TLA UI.

Declaration Syntax:

Visual Basic

```
Sub SetLockedWindows (lockedSet As ILockingWindow(), _
      referenceWindow As ILockingWindow)
```

C#

```
void SetLockedWindows (ILockingWindow[] lockedSet,
      ILockingWindow referenceWindow);
```

C++

```
void SetLockedWindows (ILockingWindow* lockedSet __gc[],
      ILockingWindow* referenceWindow);
```

Arguments:

`lockedSet` – An array of ILockingWindow Objects whose cursor positions and center screen positions will be locked together by the TLA application. This argument is allowed to be null, which indicates that no windows in the system should be locked together.

`referenceWindow –` This is an element from the lockedSet array that will serve as the reference window for the set. If this argument is not null, then all windows in the set will have their cursors and center screen positions set equal to those of the reference window. If this argument is null, then all windows in the set will retain their current cursor and center positions; and time offsets between windows will be maintained.

Return Value:

None.

Exceptions Thrown:

ArgumentNullException :
Condition: One of the array elements is a null reference.
Message: The array passed to SetLockedWindows contains a null reference.

ArgumentException :
Condition: The `referenceWindow` object is not one of the objects in the `lockedSet` `array`.
Message: The reference window must be a member of the locked set.

Condition: The array is not null and has less than two elements.
Message: The array argument passed to SetLockedWindows had less than two elements.

Remarks:

Note that the current set of locked windows can be obtained from the LockedWindows property. By using operations on collections the LockedWindows array can be modified to add or remove individual windows from the locked set.

2.2.2.23 ITlaSystem.GetInterprobeConnections

Description:

Gets an array of all the inter-probing connections that are currently defined in the system. These are the connections that appear in the System Inter-probing dialog box.

Declaration Syntax:

Visual Basic

```
Function GetInterprobeConnections () As InterprobeConnection()
```

C#

```
InterprobeConnection[] GetInterprobeConnections ();
```

C++

```
InterprobeConnection* GetInterprobeConnections () __gc [];
```

Arguments:

None.

Return Value:

An array of InterprobeConnection objects is returned. If there are no system inter-probing connections defined, then a null reference is returned.

Exceptions Thrown:

None.

Remarks:

InterprobeConnection is a value type, but it is always created on the managed heap because it contains references to managed objects.

2.2.2.24 ITlaSystem.AddInterprobeConnection

Description:

Defines a physical connection between a TLA7AXX analog probe output and a oscilloscope input channel. After defining a connection, it will appear in the System Inter-probing dialog box.

Declaration Syntax:

Visual Basic

```
Sub AddInterprobeConnection (connection As InterprobeConnection)

Sub AddInterprobeConnection (physicalLA As IPhysicalLAModule _
                             outputChannel As Int32 _
                             oscilloscope As IPhysicalModule_
                             inputChannel As Int32)
```

C#

```
void AddInterprobeConnection (InterprobeConnection connection)

void AddInterprobeConnection (IPhysicalLAModule physicalLA
                              Int32 outputChannel
                              IPhysicalModule oscilloscope
                              Int32 inputChannel)
```

C++

```
void AddInterprobeConnection (InterprobeConnection* connection)

void AddInterprobeConnection (IPhysicalLAModule* physicalLA
                              Int32 outputChannel
                              IPhysicalModule* oscilloscope
                              Int32 inputChannel)
```

Arguments:

`connection` – A connection that is defined within an InterprobeConnection object.

`physicalLA` – A reference to the physical LA module that is the source of the inter-probing signal.

`outputChannel` – The channel number of the analog probe output that is connected to the inter-probe cable. The channel number range is 1 – N, where N is the number of analog output channels on the physical LA.

`oscilloscope` – A reference to the oscilloscope, represented as a physical generic module, that is the destination of the analog signal.

`inputChannel` – The channel number of the input channel that is connected to the inter-probe cable. The channel number range is 1 – N, where N is the number of analog input channels on the oscilloscope.

Return Value:

None.

Exceptions Thrown:

ArgumentNullException:

Condition: Either the `physicalLA` or `oscilloscope` arguments are null references.
Message: <parameter-name> argument passed as null, but cannot be null.

ArgumentException:

Condition: Invalid physical LA module.
Message: <Module name> does not support inter-probing.

Condition: Invalid InterprobeConnection object passed.
Message: The InterprobeConnection object is invalid.

ArgumentOutOfRangeException:

Condition: Either in`putChannel` or `outputChannel` was not in the range of valid channel numbers for the given module.
Message: Channel number not valid.

Remarks:

The physical LA module defined as connected to the oscilloscope must be of a kind that has analog probe outputs. If it is not, then an exception will be thrown. The physical LA is defined either by the `physicalLA` argument or by a valid `connection` argument.

2.2.2.25 ITlaSystem.RemoveInterprobeConnection

Description:

Removes a system inter-probing connection between a logic analyzer module and an oscilloscope. After this method is called, the change will be reflected in the System Inter-probing dialog.

Declaration Syntax:

Visual Basic

```
Sub RemoveInterprobeConnection (connection As InterprobeConnection)

Sub RemoveInterprobeConnection (physicalLA As IPhysicalLAModule _
                                outputChannel As Int32 _
                                oscilloscope As IPhysicalModule _
                                inputChannel As Int32)
```

C#

```
void RemoveInterprobeConnection (InterprobeConnection connection)

void RemoveInterprobeConnection (IPhysicalLAModule physicalLA
                                 Int32 outputChannel
                                 IPhysicalModule oscilloscope
                                 Int32 inputChannel)
```

C++

```
void RemoveInterprobeConnection (InterprobeConnection* connection)

void RemoveInterprobeConnection (IPhysicalLAModule* physicalLA
                                 Int32 outputChannel
                                 IPhysicalModule* oscilloscope
                                 Int32 inputChannel)
```

Arguments:

`connection` – A connection that is defined within an InterprobeConnection object.

`physicalLA` – A reference to the physical LA module that is the source of the inter-probing signal.

`outputChannel` – The channel number of the analog probe output that is connected to the inter-probe cable. The channel number range is 1 – N, where N is the number of analog output channels on the physical LA.

`oscilloscope` – A reference to the oscilloscope module that is the destination of the analog signal.

`inputChannel` – The channel number of the input channel that is connected to the inter-probe cable. The channel number range is 1 – N, where N is the number of analog input channels on the oscilloscope.

Return Value:

None.

Exceptions Thrown:

ArgumentNullException:
Condition: Either the `physicalLA` or `oscilloscope` arguments are null references.
Message: <parameter-name> argument passed as null, but cannot be null.

ArgumentException:
Condition: Invalid physical LA module.
Message: <Module name> does not support inter-probing.

Condition: Invalid InterprobeConnection object passed.
Message: The InterprobeConnection object is invalid.

ArgumentOutOfRangeException:
Condition: Either in`putChannel` or `outputChannel` was not in the range of valid channel numbers for the given module.
Message: Channel number not valid.

Remarks:

If valid arguments are passed in but do not represent a current system inter-probing connection, this method will return without throwing an exception.

2.2.3 ITlaSystem Events

2.2.3.1 ITlaSystem.Defaulted

Description:

The TLA raises this event when the entire TLA system has been returned to a default state. This occurs when the user selects the Default System command from the UI and also when the ITlaSystem.Default method is called programmatically.

Declaration Syntax:

Visual Basic

```
Event Defaulted As EventHandler
```

C#

```
event EventHandler Defaulted;
```

C++

```
__event EventHandler Defaulted;
```

Event Data:

None.

Remarks:

When the system is defaulted, all collections of data sources, data windows, locked windows, and plug-ins are subject to change because all such collections will be rebuilt. Event notifications for changes to the system’s object collections, such as ITlaSystem.DataSourceChanged, are not fired. Instead the Defaulted event is raised to inform interested clients that the state of the entire ITlaSystem object has been defaulted.

2.2.3.2 ITlaSystem.DataSourcesChanged

Description:

This event is raised when the a data source is added or removed from the system.

Declaration Syntax:

Visual Basic

```
Event DataSourcesChanged As CollectionChangedHandler
```

C#

```
event CollectionChangedHandler DataSourcesChanged;
```

C++

```
__event  CollectionChangedHandler DataSourcesChanged;
```

Event Data:

Handlers of the event are passed a SystemCollectionEventArgs object that indicates which data source was affected and whether it was added or removed. Event handlers can also examine the ITlaSystem.DataSources collection to determine what data sources are currently in the system.

Remarks:

When a data source is removed from the system, this event is raised after the IDataSource object has been removed from the ITlaSystem.DataSources collection. Immediately after the event is fired, the TLA application will perform clean-up. If the removed object implements IDisposable, the application will call Dispose. The application will always set all its references to the object to null. Clients should use this event to identify whether they have references to the removed object and set those references to null so that garbage collection can free managed resources.

Some IDataSource objects can be plug-ins too. When data source plug-ins are added or removed from the system, the change is reflected both in ITlaSystem.DataSources and in ITlaSystem.PlugIns, in which case both the DataSourcesChanged and the PlugInsChanged events are raised.

This event is not raised when the system is defaulted. When the system is defaulted, the ITlaSystem.Defaulted event will be raised to indicate that all ITlaSystem collections, including ITlaSystem.DataSources has changed.

2.2.3.3 ITlaSystem.InterprobeConnectionsChanged

Description:

This event is raised when interprobe connection list has been updated and needs to be refreshed.

Declaration Syntax:

Visual Basic

```
Event InterprobeConnectionsChanged As CollectionChangedHandler
```

C#

```
event CollectionChangedHandler InterprobeConnectionsChanged;
```

C++

```
__event  CollectionChangedHandler InterprobeConnectionsChanged;
```

Event Data:

none

Remarks:

This event is raised at every instance of an Add or Remove of a interprobed connection. This should be an indication to the client to re-call the GetInterprobeConnections method to refresh it's list of connections.

2.2.3.4 ITlaSystem.DataWindowsChanged

Description:

The system raises this event when A listing, waveform, or plug-in data window is added or removed from the system.

Declaration Syntax:

Visual Basic

```
Event DataWindowsChanged As CollectionChangedHandler
```

C#

```
event CollectionChangedHandler DataWindowsChanged;
```

C++

```
__event  CollectionChangedHandler DataWindowsChanged;
```

Event Data:

Handlers of this event are passed a SystemCollectionEventArgs object that indicates which data window was affected and whether it was added or removed. The ITlaSystem.DataWindows collection can be examined to determine what listing, waveform, and plug-in data windows are currently in the system.

Remarks:

When a data window is removed from the system, this event is raised after the IDataWindow object has been removed from the ITlaSystem.DataWindows collection. Immediately after the event is fired, the TLA application will perform clean-up. If the removed object implements IDisposable, the application will call Dispose. The application will always set all its references to the object to null. Clients should use this event to identify whether they have references to the removed object and set those references to null so that garbage collection can free managed resources.

Some IDataWindow objects can be plug-ins too. When data window plug-ins are added or removed from the system, the change is reflected both in ITlaSystem.DataWindows and in ITlaSystem.PlugIns, in which case both the DataWindowsChanged and the PlugInsChanged events are raised.

This event is not raised when the system is defaulted. When the system is defaulted, the Defaulted event will be raised to indicate that all ITlaSystem collections, including ITlaSystem.DataWindows has changed.

2.2.3.5 ITlaSystem.PlugInsChanged

Description:

The system fires this event when a plug-in is added or removed from the system.

Declaration Syntax:

Visual Basic

```
Event PlugInsChanged As CollectionChangedHandler
```

C#

```
event CollectionChangedHandler PlugInsChanged;
```

C++

```
__event  CollectionChangedHandler PlugInsChanged;
```

Event Data:

Handlers of this event are passed a SystemCollectionEventArgs object that indicates which plug-in was affected and whether it was added or removed. The ITlaSystem.PlugIns collection can be examined to determine what plug-ins are currently in the system.

Remarks:

When a plug-in is removed from the system, this event is raised after the IPlugIn object has been removed from the ITlaSystem.PlugIns collection. Immediately after the event is fired, the TLA application will perform clean-up. The application will call Dispose on the plu-ins IDisposable interface, and the application will set all its references to the object to null. Clients should use this event to identify whether they have references to the removed object and set those references to null so that garbage collection can free managed resources.

Data window plug-ins will also be removed from the DataWindows collection, and data source plug-ins will also be removed from DataSources collection. The DataWindowsChanged and DataSources changed events are raised as appropriate.

This event is not raised when the system is defaulted. When the system is defaulted, the Defaulted event will be raised to indicate that all ITlaSystem collections, including ITlaSystem.PlugIns has changed.

2.2.3.6 ITlaSystem.LockedSetChanged

Description:

The system raises this event when the set of locked windows changes. Event handlers can examine the ITlaSystem.LockedWindows collection to determine which data windows are currently locked together.

Declaration Syntax:

Visual Basic

```
Event LockedSetChanged As EventHandler
```

C#

```
event EventHandler LockedSetChanged;
```

C++

```
__event  EventHandler LockedSetChanged;
```

Event Data:

Event handlers for this kind of event do not take parameters.

Remarks:

This event is fired when windows are added and replaced to the set of locked windows. The event is not fired due to changes in state in the windows of the locked set.

2.3 ISystemOptionAccess

Description:

This interface provides the ability to query System Options. Instances of these objects can be created through interface (ITlaSystem) by method (CreateSystemOptionAccess).

Namespace: Tektronix.LogicAnalyzer.Common

Declaration Syntax:

Visual Basic

```
Interface ISystemOptionAccess
```

C#

```
interface ISystemOptionAccess
```

C++

```
__gc __interface ISystemOptionAccess
```

2.3.1 ISystemOptionAccess Methods

2.3.1.1 ISystemOptionAccess.GetOption

Description:

This method returns an object of the specified option.

Declaration Syntax:

Visual Basic

```
Function GetOption (optionPageName As String, optionName As String) As
Object
```

C#

```
object GetOption (String optionPageName, String optionName)
```

C++

```
Object* GetOption (String* optionPageName, String* optionName)
```

Arguments:

optionPageName – The name of the page that the option belongs to.
OptionName – The name of the specified option.

Return Value:

Returns a reference to an object representing the requested option. This method reads registry for the requested system option. If the requested option is not listed in the registry, then the reference is null. Otherwise the option value is boxed into the returned object.

Exceptions Thrown:

None.

Examples:

C# Example

```
using System
using Tektronix.LogicAnalyzer.Common
using Tektronix.LogicAnalyzer.TpiNet

class TestGetOption
{
  static public void Test( ISystemOptionAccess iAccess,
                           String pageName,
                           String optionName )
{
      Object obj = iAccess.GetOption(pageName,optionName);
      if ( obj != null )
      {
            Console.WriteLine("Requested option = {0}, Type is {1}",
                        obj.ToString(), obj.GetType().ToString() );
      }
}
```

System Option Names Table

Option Page Name	Option Name
Color Schemes	Other Colors
	Zillions of Colors
Defaults	Radix
	TriggerTab
Preferences	NewDataWindowType
	ListingColorScheme
	ShowGlitchesListing
	ShowCompareListing
	ShowQualGaps
	DataFontSize
	HistogramDataFontSize
	HistogramColorScheme
	WaveformColorScheme
	ShowGraticule
	ShowGlitchesWaveform
	ShowCompareWaveform
	ChannelWaveformHeight
	GroupWaveformHeight
	DsoWaveformHeight
	SourceTabSpacing
	SourceLineNumbers
	SourceDataFontSize
	SourceColorScheme
	ShowStatusBar
	ShowControlBar
	ShowToolsBar
	DisplayConfirmations
	ShowSHinCustomClocking
	TrackerRectBehavior
Presets	TTL
	CMOS
	ECL
	User 1
	User 2
Startup	StartupSystemType
	SavedSystemPath
	LastOpenSystemPath

Print	Margins
Source Files	Path
	Suffix
Tools	NumTools
	MenuName#
	PathName#
	DefaultArguments#
	InitialDirectory#
	RunMinimized#

2.4 ISymbolFile

Description:

This interface provides services related to a loaded symbol file. Clients can use this interface to translate bit patterns into symbols. ISymbolFile also allows clients to request that a symbol file is not unloaded until the client releases it.

Namespace: Tektronix.LogicAnalyzer.TpiNet

Declaration Syntax:

Visual Basic

```
Interface ISymbolFile
      Inherits IDisposable
      Inherits IValidity
```

C#

```
interface ISymbolFile : IDisposable, IValidity
```

C++

```
__gc __interface ISymbolFile : public IDisposable, public IValidity
```

Remarks:

This interface inherits from IValidity, which is used to indicate whether or not an ISymbolFile object can still be used for translation. An instance of ISymbolFile provided by the TLA application will remain valid until its associated symbol file is unloaded, at which time the instance will set IsValid to false, set IsGarbage to true, and raise the ValidityChanged event. Once an ISymbolFile object becomes invalid, it never returns to a valid state. Therefore clients that have references to invalid ISymbolFile objects should remove their references to them as soon as the ValidityChanged event is raised.

This interface also inherits from IDisposable. A client that no longer needs an ISymbolFile object must call the Dispose method. Failure to do so will waste memory.

2.4.1 ISymbolFile Properties

2.4.1.1 ISymbolFile.FilePath

Description:

The value of this property is a string representing the full path of the symbol file to which an ISymbolFile object provides access.

Declaration Syntax:

Visual Basic

```
Property Readonly FilePath As String
```

C#

```
string FilePath { get; }
```

C++

```
String* get_FilePath ();
```

Arguments:

None.

Exceptions Thrown:

None.

2.4.1.2 ISymbolFile.IsPreventingUnload

Description:

Gets or sets a flag that indicates whether this ISymbolFile instance prevents the TLA application from unloading the associated symbol file. When the value is true the file cannot be unloaded by the system. When the value is false, this instance allows the file to unload (but other instances can still prevent unloading).

Declaration Syntax:

Visual Basic

```
Property IsPreventingUnload As Boolean
```

C#

```
bool IsPreventingUnload { set; get; }
```

C++

```
bool get_IsPreventingUnload ();
void set_IsPreventingUnload (bool);
```

Arguments:

None

Exceptions Thrown:

Exception:
Condition: Attempt is made to set this property to true when IsValid is false or IsGarbage is true.
Message: Cannot set value to true because the object is not in a valid state.

Remarks:

Although an instance of ISymbolFile can individually prevent its target symbol file from unloading, it cannot cause the file to unload, nor can it prevent the file from being reloaded (by calls to ITlaSystem.LoadSymbolFile). The system maintains a list of all ISymbolFile objects that target a specific symbol file. Unless the system is defaulted or the application shuts down, a symbol file cannot be unloaded until all the ISymbolFile objects on its list have their IsPreventingUnload properties set to false.

2.4.2 ISymbolFile Methods

2.4.2.1 ISymbolFile.Translate

Description:

Uses the symbol table definitions in the associated symbol file to convert a bit pattern into a string.

Declaration Syntax:

Visual Basic

```
Function Translate(value As Integer) As String

Function Translate(value As Integer, _
      maxSymbolLength As int) As String
```

C#

```
string Translate (int value)
string Translate (int value, int maxSymbolLength)
```

C++

```
String* Translate (int value)
string* Translate (int value, int maxSymbolLength)
```

Arguments:

`value` – A 32 bit value that is to be translated into a string representation.

`maxSymbolLength` – This specifies the maximum length of the string returned. When the translated string is longer than the specified length, it is truncated.

Return Value:

A reference to string object is returned. The value of the string is a symbolic representation of the bit pattern passed in.

Exceptions Thrown:

None.

Remarks:

The argument to this method is a 32 bit pattern. If a pattern symbol file is being used, then the bit pattern, not the integer value, is what gets translated. If a range symbol file is being used, then an unsigned interpretation of the bit pattern is used to translate the value into a symbol.

If the single argument overload is used, the returned string always has the full length translation of the bit pattern.

2.4.3 ISymbolFile Events

2.4.3.1 ISymbolFile.Reloaded

Description:

This event is raised when the loaded symbol file associated with an ISymbolFile object has been reloaded, usually changing the contents of the file. A symbol file can be reloaded from the user interface or through the programmatic interface.

This event is only raised when the modification date of the file has changed since it was last loaded. If the same version of the file is loaded twice, the second load does not raise the event. If the file is loaded, modified, and loaded again, then this event will be raised.

Declaration Syntax:

Visual Basic

```
Event Reloaded As EventHandler
```

C#

```
event EventHandler Reloaded;
```

C++

```
__event EventHandler* Reloaded;
```

Event Data:

A reference to the sender of the event is passed to the event handler. The EventArgs parameter is always EventArgs.Empty.

2.5 ITlaRunControl

Description:

Gives programmatic control over how the system as whole performs acquisitions.

Namespace: Tektronix.LogicAnalyzer.TpiNet

Declaration Syntax:

Visual Basic

```
Interface ITlaRunControl
```

C#

```
interface ITlaRunControl
```

C++

```
__gc __interface ITlaRunControl
```

Remarks:

This interface is used to start and stop acquisitions, as well as provide synchronous and asynchronous information about start/stop events.

2.5.1 ITlaRunControl Properties

2.5.1.1 ITlaRunControl.IsRepetitive

Description:

Gets or sets a Boolean indication whether or not the TLA is in repetitive acquisition mode.

Declaration Syntax:

Visual Basic

```
Property IsRepetitive As Boolean
```

C#

```
bool IsRepetitive { set; get;}
```

C++

```
bool get_IsRepetitive ();
void set_IsRepetitive (bool);
```

Arguments:

None.

Exceptions Thrown:

None.

2.5.1.2 ITlaRunControl.RepetitiveProperties

Description:

Sets and gets the properties of a repetitive a acquisition. The value of this property is an object of type RepetitiveSetup, which is used to define repetitive acquisition settings. This property affects acquisitions only when the IsRepetitive property is set to true.

Declaration Syntax:

Visual Basic

```
Property RepetitiveProperties As RepetitiveSetup
```

C#

```
RepetitiveSetup RepetitiveProperties { set; get; }
```

C++

```
RepetitiveSetup get_Property-Name ();
void set_RepetitiveProperties (RepetitiveSetup);
```

Arguments:

None

Exceptions Thrown:

ArgumentException:
Condition: A RepetiveSetup object with invalid settings have been used to set this property.
Message: Attempt to set RepetitiveProperties with invalid settings.

Remarks:

To avoid an ArgumentException being thrown when setting this property, the method RepetitiveSetup.IsValid can be called to check the validity of the setup.

2.5.1.3 ITlaRunControl.IsRunning

Description:

Gets a Boolean indication of whether the system is currently running. A true value indicates the system is performing an acquisition. A false value indicates the system is idle.

Declaration Syntax:

Visual Basic

```
ReadOnly Property IsRunning As Boolean
```

C#

```
bool IsRunning { get; }
```

C++

```
bool get_IsRunning ();
```

Arguments:

None.

Exceptions Thrown:

This property does not throw exceptions.

Remarks:

This property can be used to check whether the system is running or idle. Clients that need notification of changes in run state can subscribe to ITlaRunControl events that are fired when the system run status changes.

2.5.1.4 ITlaRunControl.RunCount

Description:

Gets the number of acquisitions performed during the current session of the TLA application. All iterative acquisitions that occur during a repetitive acquisition are counted together as one run.

Declaration Syntax:

Visual Basic

```
ReadOnly Property RunCount As Integer
```

C#

```
int RunCount { get; }
```

C++

```
int get_RunCount ();
```

Arguments:

None.

Exceptions Thrown:

None.

2.5.2 ITlaRunControl Methods

2.5.2.1 ITlaRunControl.Run

Description:

Starts a new acquisition using the current setup of the system and its data sources.

Declaration Syntax:

Visual Basic

```
Sub Run ()
```

C#

```
void Run ()
```

C++

```
void Run ()
```

Arguments:

None.

Return Value:

None.

Exceptions Thrown:

ApplicationException :

Condition: The system has no enabled data sources that are capable of acquisition.
Message: There are no enabled acquisition data sources.

Condition: The system cannot start due to a diagnostic failure.
Message: A diagnostic failure prevented acquisition start-up.

Condition: The system cannot start due to an iView failure, such as a disconnected iView cable..
Message: An iView failure prevented acquisition start-up.

Remarks:

The system must have at least one enabled data source that is capable of acquiring data. The acquisition data source can be an internal LA or DSO module, an external oscilloscope, or a data source plug-in that can acquire data.

The Run method attempts to start an acquisition and then returns. If the system cannot even begin an acquisition, the method will throw an exception. Otherwise, clients can be asynchronously notified of run completion by subscribing to the RunComplete event. Clients only interested in repetitive acquisitions can subscribe to RepetitiveRunComplete.

2.5.2.2 ITlaRunControl.Stop

Description:

Stops the currently running acquisition.

Declaration Syntax:

Visual Basic

```
Sub Stop ()
```

C#

```
void Stop ()
```

C++

```
void Stop ()
```

Arguments:

None.

Return Value:

None

Exceptions Thrown:

None.

Remarks:

This method instructs the system and all its acquiring data sources to stop acquisition. The system is not necessarily stopped by the time the method returns. Clients that need to know when acquisitions are complete should subscribe to the RunComplete event. Clients only interested in repetitive acquisitions can subscribe to RepetitiveRunComplete.

If the system is not running when Stop is called, then nothing happens. Calling Stop when the system is idle will not raise a RunComplete event.

2.5.2.3 ITlaRunControl.GetRepetitiveStopReason

Description:

Indicates why the last repetitive acquisition stopped.

Declaration Syntax:

Visual Basic

```
Function GetRepetitiveStopReason () As RepetitiveStopReason
```

C#

```
RepetitiveStopReason GetRepetitiveStopReason ();
```

C++

```
RepetitiveStopReason GetRepetitiveStopReason ();
```

Arguments:

None.

Return Value:

Returns a member of the RepetitiveStopReason enumeration.

Exceptions Thrown:

This method does not throw exceptions.

Remarks:

This method can be called at any time, but is most meaningful in the handler of a ITlaRunControl.RunCompleted event. If no repetitive acquisitions have stopped since the application was launched, then RepetitiveStopReason.UnknownReason is returned.

2.5.3 ITlaRunControl Events

2.5.3.1 ITlaRunControl.RunStarted

Description:

The TLA fires this event whenever the system run state changes from idle to running.

Declaration Syntax:

Visual Basic

```
Event RunStarted As EventHandlder
```

C#

```
event EventHandlder RunStarted;
```

C++

```
__event EventHandlder RunStarted;
```

Event Data:

None.

Remarks:

This event is only fired when an acquisition has successfully started. If an attempt to start the system fails, then this event is not fired. For example, if the system has no enabled data sources capable of acquisition, a call to ITlaRunControl.Run will throw an exception and the system will remain idle.

2.5.3.2 ITlaRunControl.RunCompleted

Description:

The TLA fires this event whenever the system run state changes from running to idle.

Declaration Syntax:

Visual Basic

```
Event RunCompleted As EventHandler
```

C#

```
event EventHandler RunCompleted;
```

C++

```
__event EventHandler RunCompleted;
```

Event Data:

None.

Remarks:

This event is raised when any kind of acquisition has completed, both single-shot and repetitive. For repetitive acquisitions, this event is fired only when the run is completely stopped. The event is not fired when iterations of a repetitive run complete.

2.5.3.3 ITlaRunControl.RepetitiveIterationStopped

Description:

This event is fired when an iteration of a repetitive run has completed, but the system is not idle. This allows clients to process data or update state between iterations of a repetitive acquisition.

Declaration Syntax:

Visual Basic

```
Event RepetitiveIterationStopped As EventHandler
```

C#

```
event EventHandler RepetitiveIterationStopped;
```

C++

```
__event EventHandler RepetitiveIterationStopped;
```

Event Data:

None.

Remarks:

Plug-in clients that need to do significant processing between repetitive acquisitions should not do so within the handler for this event. Doing so will make the TLA UI unresponsive. Instead plug-in objects can subscribe to the event ITlaPlugInSupport.SystemIdle. This will provide support for idle time processing until processing is complete.

2.5.3.4 ITlaRunControl.RepetiveRunCompleted

Description:

The TLA application fires this event when a repetitive acquisition has completed, and the system has entered the idle state.

Declaration Syntax:

Visual Basic

```
Event RepetitiveRunStopped As RepetitiveStopHandler
```

C#

```
event RepetitiveStopHandler RepetitiveRunStopped;
```

C++

```
__event RepetitiveStopHandler RepetitiveRunStopped;
```

Event Data:

The event handler is passed a member of the RepetitiveStopReason enumeration, which indicates the reason why the run stopped.

Remarks:

Clients can subscribe to this event if they need to distinguish between the completion of single-shot acquisitions and repetitive ones. For clients interested in repetitive acquisitions, this event also passes a repetitive stop reason to event handlers.

This event is never raised after a single-shot acquisition.

2.6 IListingWindow

Description:

IListingWindow objects represent the listing windows that are a standard part of the TLA user interface.

Namespace: Tektronix.LogicAnalyzer.TpiNet

Declaration Syntax:

Visual Basic

```
Interface IListingWindow
      Inherits ISearchableWindow
      Inherits IExportData
      Inherits IDefault
```

C#

```
interface IListingWindow : ISearchableWindow, IExportData, IDefault
```

C++

```
__gc __interface IListingWindow
      : ISearchableWindow, IExportData, IDefault
```

Remarks:

IListingWindow inherits from ISearchableWindow, which means that it has the ability to be locked to other data windows and clients can programmatically perform any search that is defined within the window.

IListingWindow does not provide a means to add or edit search definitions. Loading a system will also load all data windows and search definitions that were present when the system was saved. To use a search definition, a system must be loaded that has a data window containing the desired search.

2.6.1 IListingWindow Methods

2.6.1.1 IListingWindow.Default

Description:

Defaulting a listing window updates which columns are shown and puts columns in their default order. Calling this method has the same effect as using the Default Columns command in the Edit menu.

Declaration Syntax:

Visual Basic

```
Sub Default ()
```

C#

```
void Default ()
```

C++

```
void Default ()
```

Arguments:

None.

Return Value:

None.

Exceptions Thrown:

TlaProhibitedDuringRunException :
Condition: Attempt made to default the window during an acquisition.
Message: Cannot default window during acquisition.

Remarks:

Listing windows can be defaulted only when the system is not running. Clients should always catch the exception that is thrown when Default is called during an acquisition.

2.6.2 IListingWindow Events

2.6.2.1 IListingWindow.Defaulted

Description:

This event is raised when a listing window is defaulted.

Declaration Syntax:

Visual Basic

```
Event Defaulted As EventHandler
```

C#

```
event EventHandler Defaulted;
```

C++

```
__event EventHandler Defaulted;
```

Event Data:

Event handlers receive a reference to the data window that was defaulted.

Remarks:

When the system is defaulted, listing windows are not defaulted. They are destroyed instead..

2.6.2.2 IListingWindow.Export

Description:

This method exports the data in a listing window to a specified file.

Declaration Syntax:

Visual Basic

```
Sub Export (file As String, args As Ojbect)
```

C#

```
void Export (string file, object args)
```

C++

```
void Export (String* file, Object* args)
```

Arguments:

`file` – The full path of the file that will contain the exported data.

`args` – This argument is not used at this time.

Return Value:

None.

Exceptions Thrown:

ArgumentNullException :
Condition: The file argument is null.
Message: Null argument provided to Export.

ArgumentException :
Condition: The file argument does not represent a valid path.
Message: The file name or path is invalid.

PathTooLongException :
Condition: The file path contained in `filename` is too long.
Message: The passed-in file name is too long.

Remarks:

Since many problems can occur during file operations, the Export method should always be called within a try block, so that exceptions can be caught. The above list of possible exceptions cannot be guaranteed complete, so a catch all clause can be used in addition to catching specific exceptions of interest.

2.7 IWaveformWindow

Description:

IWaveformWindow objects represent the waveform windows that are a standard part of the TLA user interface.

Namespace: Tektronix.LogicAnalyzer.TpiNet

Declaration Syntax:

Visual Basic

```
Interface IWaveformWindow
      Inherits ISearchableWindow
      Inherits IDefault
```

C#

```
interface IWaveformWindow : ISearchableWindow, IDefault
```

C++

```
__gc __interface IWaveformWindow
      : public ISearchableWindow, public IDefault
```

Remarks:

IWaveformWindow inherits from ISearchableWindow, which means that it has the ability to be locked to other data windows and clients can programmatically perform searches on any search that has defined within the window.

IWaveformWindow does not provide a means to add or edit search definitions. Loading a system will also load all data windows and search definitions that were present when the system was saved. To use a search definition, a system must be loaded that has a data window containing the desired search.

2.7.1 IWaveformWindow Methods

2.7.1.1 IWaveformWindow.Default

Description:

Defaulting a waveform window updates which waveforms are shown and puts waveforms in their default order. Calling this method has the same effect as the Default Waveforms command in the Edit menu.

Declaration Syntax:

Visual Basic

```
Sub Default ()
```

C#

```
void Default ()
```

C++

```
void Default ()
```

Arguments:

None.

Return Value:

None.

Exceptions Thrown:

TlaProhibitedDuringRunException :
Condition: Attempt made to default the window during an acquisition.
Message: Cannot default waveform window during acquisition.

Remarks:

Waveform windows can be defaulted only when the system is not running. Clients should always catch the exception that is thrown when Default is called during an acquisition.

2.7.2 IWaveformWindow Events

2.7.2.1 IWaveformWindow.Defaulted

Description:

This event is raised when a waveform window is defaulted.

Declaration Syntax:

Visual Basic

```
Event Defaulted As EventHandler
```

C#

```
event EventHandler Defaulted;
```

C++

```
__event EventHandler Defaulted;
```

Event Data:

Event handlers receive a reference to the data window that was defaulted.

Remarks:

When the system is defaulted, listing waveform are not defaulted. They are destroyed instead..

2.8 ISaveModule

Description:

Module objects in the system that can be independently saved implement this interface. ISaveModule provides a means to save the module and its data to file and to provide notification when this occurs.

Namespace: Tektronix.LogicAnalyzer.Common

Declaration Syntax:

Visual Basic

```
Interface ISaveModule
```

C#

```
interface ISaveModule
```

C++

```
__gc __interface ISaveModule
```

2.8.1 ISaveModule Methods

2.8.1.1 ISaveModule.SaveModule

Description:

When an object implements ISaveModule, this method causes the object to save itself to a file.

Declaration Syntax:

Visual Basic

```
Sub SaveModule ( filename As String, comment As String, data As
SaveDataOptions )
```

C#

```
void SaveModule ( String filename, String comment, SaveDataOptions data
)
```

C++

```
void SaveModule ( String* filename, String* comment, SaveDataOptions
data )
```

Arguments:

`fileName` – The full path of the file to be saved.

`comment` – A user comment to be save along with the file. This comment can be seen in Load Module dialog in the application UI. This value can be either a null reference or an empty string.

data – a member of the SaveData enumeration that determines whether module data is saved to the file.

Return Value:

None.

Exceptions Thrown:

TlaFileOperationException :
Condition: The system was running during the save operation.
Message: The file operation could not complete because the system was running.

ArgumentNullException :
Condition: The `fileName` argument is null.
Message: Null argument passed to SaveModule.

ArgumentException :
Condition: The `fileName` argument is not a valid filename.
Message: Invalid file name passed to SaveModule.

PathTooLongException :

Condition: The file path contained in `filename` is too long.
Message: The passed-in file name is too long.

UnauthorizedAccesException :

Condition: The specified file is read-only.
Message: The specified file is read-only.

Remarks:

Since many problems can occur during file operations, the SaveModule method should always be called within a try block, so that exceptions can be caught. The above list of possible exceptions cannot be guaranteed complete, so a catchall clause can be used in addition to catching specific exceptions of interest.

2.8.2 ISaveModule Events

2.8.2.1 ISaveModule.ModuleSaved

Description:

An object that implements ISaveModule raises this event when the object has been saved. Generally an object is saved either through the user interface or by calling the SaveModule method.

Declaration Syntax:

Visual Basic

```
Event ModuleSaved As EventHandler
```

C#

```
event EventHandler ModuleSaved;
```

C++

```
__event EventHandler ModuleSaved;
```

Event Data:

None.

2.9 ILoadModule

Description:

Module objects in the system that can be independently loaded implement this interface. ILoadModule provides a means to load the module from file and to provide notification when this occurs. This operation is equivalent to choosing “Load Module” in the UI from the file menu.

Namespace: Tektronix.LogicAnalyzer.Common

Declaration Syntax:

Visual Basic

```
Interface ILoadModule
```

C#

```
interface ILoadModule
```

C++

```
__gc __interface ILoadModule
```

2.9.1 ILoadModule Methods

2.9.1.1 ILoadModule.LoadModule

Description:

Loads the module in the system with setup information from the file.

Declaration Syntax:

Visual Basic

```
Sub LoadModule (fileName As String, dataSourceName As String)
```

C#

```
void LoadModule (string fileName, string dataSourceName)
```

C++

```
void LoadModule (String* fileName, String* dataSourceName)
```

Arguments:

`fileName` – The full path of the file that contains the saved data source.

`dataSourceName` – This is name of the module or the name of a data source plug-in that is saved in the file.

Return Value:

None.

Exceptions Thrown:

TlaNoDataException :
Condition: The specified module does not have any data saved in the file.
Message: Module <dataSourceName> does not have any saved data.

TlaFileOperationException :
Condition: Several file problems, not directly related to the file system, can cause the TLA to throw this exception. The exception can be examined to determine what problem was at fault.
Message: <The message depends on the failure reason.>

TlaUnsupportedSetupException :
Condition: The `dataSourceName` argument refers to an incompatible module.
Message: Module <dataSourceName> type is incompatible for LoadModule.

ArgumentNullException :
Condition: The `filename` argument is null.
Message: Null argument passed to LoadModule.

ArgumentException :

  Condition: The `filename` argument is not a valid filename.
  Message: Invalid file name passed to LoadModule.

PathTooLongException :

  Condition: The file path contained in `fileName` is too long.
  Message: The passed-in file name is too long.

Remarks:

Since many problems can occur during file operations, the LoadModule method should always be called within a try block so exceptions can be caught. The above list of possible exceptions cannot be guaranteed complete, so a catchall clause can be used in addition to catching specific exceptions of interest.

2.9.2 ILoadModule Events

2.9.2.1 ILoadModule.ModuleLoaded

Description:

An object that implements ILoadModule raises this event when the object has been loaded. Generally an object is loaded either through the user interface or by calling the LoadModule method.

Declaration Syntax:

Visual Basic

```
Event ModuleLoaded As EventHandler
```

C#

```
event EventHandler ModuleLoaded;
```

C++

```
__event EventHandler ModuleLoaded;
```

Event Data:

None.

2.10 IPhysicalModule

Description:

This interface describes the characteristics of a module that is physically installed in the system.

Namespace: Tektronix.LogicAnalyzer.TpiNet

Declaration Syntax:

Visual Basic

```
Interface IPhysicalModule
```

C#

```
interface IPhysicalModule
```

C++

```
__gc __interface IPhysicalModule
```

2.10.1 IPhysicalModule Properties

2.10.1.1 IPhysicalModule.ModelName

Description:

The model name of the data source as it should appear to users. Examples are “TLA7AB4” and "TDS7404."

Declaration Syntax:

Visual Basic

```
ReadOnly Property ModelName As String
```

C#

```
string ModelName { get;}
```

C++

```
String* get_ModelName ();
```

Arguments:

None.

Exceptions Thrown:

None.

2.10.1.2 IPhysicalModule.FirmwareVersion

Description:

The value of this property is a string that represents the version of firmware install in the physical module. The string usually has the form “Major.Minor.Build”, for example, “4.3.75”. However the number of dot separated fields in the string can differ depending on the how the instrument reports its firmware version and whether the TLA application is running in TLA Vu mode.

Declaration Syntax:

Visual Basic

```
ReadOnly property FirmwareVersion As String
```

C#

```
string FirmwareVersion { get; }
```

C++

```
String* get_FirmwareVersion ();
```

Arguments:

None.

Exceptions Thrown:

None.

Remarks:

If the TLA cannot determine the firmware version of a module, then an empty string is returned.

2.10.1.3 IPhysicalModule.LogicalModule

Description:

Gets a reference to the logical of which this module is a part. The logical module is an object of type ILAModule, IDSOModule, or IExternalOscilloscopeModule, depending on the type of physical module. If modules are merged, then more than one physical module will reference the same logical ILAModule object.

Declaration Syntax:

Visual Basic

```
ReadOnly Property LogicalModule As IModule
```

C#

```
IModule LogicalModule { get; }
```

C++

```
IModule* get_LogicalModule ();
```

Arguments:

None.

Exceptions Thrown:

None.

2.11 IPhysicalInternalModule

Description:

This interface represents a physical module that is physically installed inside a TLA. This interface extends the base IPhysicalModule with information about slot and mainframe number.

Namespace: Tektronix.LogicAnalyzer.TpiNet

Declaration Syntax:

Visual Basic

```
Interface IPhysicalInternalModule
      Inherits IPhysicalModule
```

C#

```
interface IPhysicalInternalModule: IPhysicalModule
```

C++

```
__gc __interface IPhysicalInternalModule : IPhysicalModule
```

2.11.1 IPhysicalInternalModule Properties

2.11.1.1 IPhysicalInternalModule.Slot

Description:

Gets the slot number within the mainframe into which the physical module has been installed.

Declaration Syntax:

Visual Basic

```
ReadOnly Property Slot As Integer
```

C#

```
int Slot { get; }
```

C++

```
int get_Slot ();
```

Arguments:

None.

Exceptions Thrown:

None.

Remarks:

The slot number is relative to the mainframe in which the physical module has been installed. The mainframe number of physical module is available through the property IPhysicalModule.MainframeNumber.

2.11.1.2 IPhysicalInternalModule.MainframeNumber

Description:

Gets the mainframe number into which the physical module is installed.

Declaration Syntax:

Visual Basic

```
Readonly Property MainframeNumber As Integer
```

C#

```
int MainframeNumber { get; }
```

C++

```
int get_MainframeNumber ();
```

Arguments:

None.

Exceptions Thrown:

None

Remarks:

Mainframes are numbered as follows: The mainframe containing the controller is number 0, the first expansion mainframe is number 1, and so on.

2.12 IPhysicalExternalOscilloscopeModule

The IPhysicalExternalOscilloscopeModule interface is identical to the IPhysicalModule interface. By using a different name to distinguish this type from the base type, the interface can be augmented with external scope specific members at a later time without affecting the physical LA or physical DSO interfaces.

2.13 IPhysicalDSOModule

The IPhysicalDSOModule interface is identical to the IPhysicalInternalModule interface. By distinguishing this type from the base type with a different name, the interface can be augmented with DSO specific members at a later time without affecting the physical LA or external scope interfaces.

2.14 IPhysicalLAModule

Description:

This interface represents an LA module that has been installed into one of the mainframes of the TLA instrument.

Namespace: Tektronix.LogicAnalyzer.TpiNet

Declaration Syntax:

Visual Basic

```
Interface IPhysicalLAModule
```

C#

```
interface IPhysicalLAModule
```

C++

```
__gc __interface IPhysicalLAModule
```

Remarks:

The IPhysicalLAModule interface is used to model functionality that is specific to a single physical logic analyzer module, as opposed to a merged set of modules. System inter-probing and analog probe outputs are dealt with on the basis of single physical modules.

2.14.1 IPhysicalLAModule Properties

2.14.1.1 IPhysicalLAModule.NumberAnalogOutputs

Description:

Gets the number of analog probe outputs on the physical logic analyzer module. Not all physical modules have analog probe outputs. If the module does not analog outputs, the value of this property is zero.

Declaration Syntax:

Visual Basic

```
ReadOnly Property NumberAnalogOutputs As Integer
```

C#

```
int NumberAnalogOutputs { get; }
```

C++

```
int get_NumberAnalogOutputs ();
```

Arguments:

None.

Exceptions Thrown:

None.

Remarks:

If the value of this property is zero, then any calls to IPhysicalLAModule.SetAnalogFeed will throw an exception.

2.14.2 IPhysicalLAModule Methods

2.14.2.1 IPhysicalLAModule.SetAnalogFeed

Description:

Sets the probe channel signal that is assigned to feed an analog output channel.

Declaration Syntax:

Visual Basic

```
Sub SetAnalogFeed (feed As IChannelBit, _
                   outputChannel As Integer)

Sub SetAnalogFeed (feedName As String, _
                   outputChannel As Integer)
```

C#

```
void SetAnalogFeed (IChannelBit feed, int outputChannel)

void SetAnalogFeed (string feedName, int outputChannel)
```

C++

```
void SetAnalogFeed (IChannelBit* feed, int outputChannel)

void SetAnalogFeed (String* feedName, int outputChannel)
```

Arguments:

`feed` – An IChannelBit object that corresponds the probe channel that is to become the feed for the analog output channel specified by `outputChannel`.

`feedName` – This string specifies the name of the probe channel that is to become the feed for the analog output channel specified by `outputChannel`. `feedName` can refer to the user name or the hardware pod name followed by the channel number enclosed in parentheses, (e.g. A0(1), A0(2), etc.).

`outputChannel` – The analog output channel whose signal source is being set. The valid range for this argument is 1 to NumberAnalogOutputs.

Return Value:

None

Exceptions Thrown:

ArgumentException:

Condition: Either the feed or feedName argument does not represent a valid channel on the physical module.

Message: Invalid probe channel given.

ArgumentOutOfRangeException:

Condition: The value of outputChannel is not in the valid range.
Message: Output channel specified out of range.

NotSupportedException:

Condition: The method is called on an IPhysicalLAModule object whose NumberAnalogOutputs value is zero.
Message: Cannot set analog feed on module without analog outputs.

Remarks:

The SetAnalogFeed method can be used along with ITlaSystem.SetInterprobeConnection to fully specify the path of signal from the LA probe tip to the input channel of an oscilloscope in the system.

Note that an analog feed can be specified even if analog output is not part of a system inter-probing connection.

2.14.2.2 IPhysicalLAModule.GetAnalogFeed

Description:

Gets the probe channel that is currently feeding a specified analog output channel.

Declaration Syntax:

Visual Basic

```
Function GetAnalogFeed (outputChannel As Integer) As IChannelBit
```

C#

```
IChannelBit GetAnalogFeed (int outputChannel)
```

C++

```
IChannelBit* GetAnalogFeed (int outputChannel)
```

Arguments:

`outputChannel` – Specifies the channel number for the output channel of interest. The value of this argument must be an integer in the range 1 to NumberAnalogOutputs.

Return Value:

Returns the IChannelBit object that represents the probe channel that is currently feeding the specified analog output channel. Clients can use this object to determine both the user name and the hardware name of the channel from the returned object.

Exceptions Thrown:

ArgumentOutOfRangeException:
Condition: The value of outputChannel is not in the valid range.
Message: Output channel specified out of range.

NotSupportedException:
Condition: The method is called on an IPhysicalLAModule object whose NumberAnalogOutputs value is zero.
Message: Cannot get analog feed on module without analog outputs.

2.15 IModule

Description:

This is the base interface for all of the physical instruments that can be installed into the LA. DSO, external oscilloscope, and LA modules have interfaces derived from this base. Almost all of the IModule interface is inherited from a collection several base interfaces.

Namespace: Tektronix.LogicAnalyzer.TpiNet

Declaration Syntax:

Visual Basic

```
Interface IModule Inherits IStandardDataSource, IModuleRunStatus,
IDefault, ISaveModule, ILoadModule
```

C#

```
interface IModule
      : IStandardDataSource, ImoduleRunStatus, IDefault, ISaveModule,
ILoadModule
```

C++

```
__gc __interface IModule
      : public IStandardDataSource, ImoduleRunStatus, IDefault,
ISaveModule, ILoadModule
```

Remarks:

The IStandardDataSource.GetAcquisitionDataObject is defined to return a reference to an IAcquisitionData object. This may be cast to an ILAAcquisitionData, an IScopeAcquisitionData object as appropriate, depending on whether the object was obtained from an ILAModule object or one of the oscilloscope objects (IDSOmodule or IExternalOscilloscope).

2.15.1 IModule Properties

2.15.1.1 IModule.PhysicalModuleArray

Description:

The value of this property is an array of IPhysicalModule objects. If the module is unmerged, the array has only one element. If the module is composed of merged LA modules, then the array contains one IPhysicalLAModule for each physical logic analyzer in the merged set, arranged in physical order, from lowest slot number to highest.

Declaration Syntax:

Visual Basic

```
ReadOnly Property PhysicalModuleArray As IPhysicalModule ()
```

C#

```
IPhysicalModule [] PhysicalModuleArray { get; }
```

C++

```
IPhysicalModule* get_PhysicalModuleArray () [];
```

Arguments:

None.

Exceptions Thrown:

None.

Remarks:

Depending on the object that implements the property, the type of elements in the array might be derived from IPhysicalModule. To obtain the specialized information in IPhysicalModule-derived objects, cast the element to the appropriate type.

IExternalOscilloscopeModule.PhysicalModuleArray is an array of IPhysicalExternalOscilloscopeModule.

IDSOModule.PhysicalModuleArray is an array of IPhysicalDSOModule.

ILAModule.PhysicalModuleArray is an array of IPhysicalLAModule.

2.16 IDSOModule

The IDSOModule interface is identical to the IModule interface. The name of the derived interface can be used to distinguish between an LA, an internal DSO, and an external oscilloscope.

2.17 IExternalOscilloscopeModule

The IExternalOscilloscopeModule interface is identical to the IModule interface. The name of the derived interface can be used to distinguish between an LA, an internal DSO, and an external oscilloscope.

2.18 ILAModule

Description:

This interface inherits from IModule. It also provides methods for getting LA specific setup objects.

Namespace: Tektronix.LogicAnalyzer.Common

Declaration Syntax:

Visual Basic

```
Interface ILAModule
      Inherits IModule
```

C#

```
interface ILAModule : IModule
```

C++

```
__gc __interface ILAModule : public IModule
```

Remarks:

The IStandardDataSource.GetAcquisitionDataObject is defined to return a reference to an IAcquisitionData object. This may be cast to an ILAAcquisitionData object in the case of ILAModule.

2.18.1 ILAModule Methods

2.18.1.1 ILAModule.GetTriggerObject

Description:

This method is used to obtain an ILATrigger object.

Declaration Syntax:

Visual Basic

```
Function GetTriggerObject () As ILATrigger
```

C#

```
ILATrigger GetTriggerObject ()
```

C++

```
ILATrigger* GetTriggerObject ()
```

Arguments:

None.

Return Value:

A reference to an object of type ILATrigger is returned.

Exceptions Thrown:

Exception-Class-Name :
    Condition: Condition under which the exception is thrown.
    Message: Message string associated with exception.

2.18.1.2 ILAModule.GetFiltersObject

Description:

This method is used to obtain an IFilters object.

Declaration Syntax:

Visual Basic

```
Function GetFiltersObject () As IFilters
```

C#

```
IFilters GetFiltersObject ()
```

C++

```
IFilters* GetFiltersObject ()
```

Arguments:

None.

Return Value:

A reference to an object of type IFilters is returned.

Exceptions Thrown:

Exception-Class-Name :
    Condition: Condition under which the exception is thrown.
    Message: Message string associated with exception.

2.18.1.3 ILAModule.GetThresholdsObject

Description:

This method is used to obtain an IThresholds object.

Declaration Syntax:

Visual Basic

```
Function GetThresholdsObject () As IThresholds
```

C#

```
IThresholds GetThresholdsObject ()
```

C++

```
IThresholds* GetThresholdsObject ()
```

Arguments:

None.

Return Value:

A reference to an object of type IThresholds is returned.

Exceptions Thrown:

Exception-Class-Name :
Condition: Condition under which the exception is thrown.
Message: Message string associated with exception.

2.18.1.4 ILAModule.GetCompareSetupObject

Description:

This method is used to obtain an ICompareSetup object.

Declaration Syntax:

Visual Basic

```
Function GetCompareSetupObject () As ICompareSetup
```

C#

```
ICompareSetup GetCompareSetupObject ()
```

C++

```
ICompareSetup* GetCompareSetupObject ()
```

Arguments:

None.

Return Value:

A reference to an object of type ICompareSetup is returned.

Exceptions Thrown:

Exception-Class-Name :
    Condition: Condition under which the exception is thrown.
    Message: Message string associated with exception.

2.18.1.5 ILAModule.GetMemoryDepthObject

Description:

This method is used to obtain an IMemoryDepth object for the LA main memory depth.

Declaration Syntax:

Visual Basic

```
Function GetMemoryDepthObject () As IMemoryDepth
```

C#

```
IMemoryDepth GetMemoryDepthObject ()
```

C++

```
IMemoryDepth* GetMemoryDepthObject ()
```

Arguments:

None.

Return Value:

A reference to an object of type IMemoryDepth is returned.

Exceptions Thrown:

Exception-Class-Name :
 Condition: Condition under which the exception is thrown.
 Message: Message string associated with exception.

Remarks:

The MagniVu memory depth is handled in a different manner by the ILATrigger interface.

2.18.1.6 ILAModule.GetAcquireModeObject

Description:

This method is used to obtain an IAcquireMode object.

Declaration Syntax:

Visual Basic

```
Function GetAcquireModeObject () As IAcquireMode
```

C#

```
IAcquireMode GetAcquireModeObject ()
```

C++

```
IAcquireMode* GetAcquireModeObject ()
```

Arguments:

None.

Return Value:

A reference to an object of type IAcquireMode is returned.

Exceptions Thrown:

None.

2.18.1.7 ILAModule.GetSamplePeriodObject

Description:

This method is used to obtain an ISamplePeriod object for the main memory sample period.

Declaration Syntax:

Visual Basic

```
Function GetSamplePeriodObject () As ISamplePeriod
```

C#

```
ISamplePeriod GetSamplePeriodObject ()
```

C++

```
ISamplePeriod* GetSamplePeriodObject ()
```

Arguments:

None.

Return Value:

A reference to an object of type ISamplePeriod is returned.

Exceptions Thrown:

Exception-Class-Name :
 Condition: Condition under which the exception is thrown.
 Message: Message string associated with exception.

Remarks:

The MagniVu sample period is handled in a different manner by the ILATrigger object.

2.18.1.8 ILAModule.GetClockModeObject

Description:

This method is used to obtain an IClockMode object.

Declaration Syntax:

Visual Basic

```
Function GetClockModeObject () As IClockMode
```

C#

```
IClockMode GetClockModeObject ()
```

C++

```
IClockMode* GetClockModeObject ()
```

Arguments:

None.

Return Value:

A reference to an object of type IClockMode is returned.

Exceptions Thrown:

None.

2.18.1.9 ILAModule.LoadSupportPackage

Description:

Loads the support package that has a given name.

Declaration Syntax:

Visual Basic

```
Sub LoadSupportPackage (name As String)
```

C#

```
void LoadSupportPackage (string name)
```

C++

```
void LoadSupportPackage (String* name)
```

Arguments:

`name` – A string object that contains the simple name of the support package to be loaded. This should be the name that appears in the Load Support Package dialog box in the TLA application user interface.

Return Value:

None.

Exceptions Thrown:

ArgumentNullException :
Condition: The `name` argument is null.
Message: Null argument passed to LoadSupportPackage.

ArgumentException :
Condition: The `name` argument is not the name of any installed support package.
Message: Invalid file name passed to LoadSupportPackage.

TlaFileOperationException :
Condition: The file was found, but the load didn't succeed because the support package requires a different module configuration. When this occurs the TlaFileOperationException.Failure member will be set to `HardwareMismatch`.
Message: The symbol file does not have a valid or recognizable format.

Condition: The file was found, but the load operation could not be completed because the file is corrupt. Under this condition, the TlaFileOperationException.Failure member will be set to `InvalidFile`.
Message: The symbol file does not have a valid or recognizable format.

PathTooLongException :
Condition: The file path contained in `filename` is too long.
Message: The passed-in file name is too long.

2.18.1.10 ILAModule.GetSupportPackageName

Description:

Returns a string reference that indicates the name of the currently loaded support package, if any.

Declaration Syntax:

Visual Basic

```
Function GetSupportPackageName () As String
```

C#

```
string GetSupportPackageName ()
```

C++

```
String* GetSupportPackageName ()
```

Arguments:

None.

Return Value:

If the LA module currently has a support package loaded, the returned string contains the name of the support package as it would appear in the Load Support Package dialog. If a support package is not loaded, then the returned string is empty.

Exceptions Thrown:

None.

Remarks:

Because the empty string is returned when no support package is loaded, this method can also be used to check for the presence of a support package.

2.18.1.11 ILAModule.GetSupportPackagePath

Description:

If the module has a loaded support package, this method returns the full path of the support package directory.

Declaration Syntax:

Visual Basic

```
Function GetSupportPackagePath () As String
```

C#

```
string GetSupportPackagePath()
```

C++

```
String* GetSupportPackagePath()
```

Arguments:

None.

Return Value:

A String is returned that represents the full path of the directory from which the current support package was loaded. If no support package is loaded, then String.Empty is returned.

Exceptions Thrown:

None.

Remarks:

When a support package is loaded, a form of redirection can occur, allowing the support package to be loaded from a different directory than indicated by the support package name. GetSupportPackagePath returns the actual directory of the loaded support package after any redirection has occurred.

2.18.1.12 ILAModule.GetSupportPackageGroupRadix

Description:

When an LA module has a support package loaded, this method returns the default radix that the support package uses with a given group.

Declaration Syntax:

Visual Basic

```
Function GetSupportPackageGroupRadix _
      (group As IChannelGroup) As GroupRadix
```

C#

```
GroupRadix GetSupportPackageGroupRadix (IChannelGroup group)
```

C++

```
GroupRadix GetSupportPackageGroupRadix (IChannelGroup* group)
```

Arguments:

`group` – Specifies the channel group for which the default support package radix is requested.

Return Value:

A member of the `GroupRadix` enumeration is returned. If a loaded support package defines a default radix for the specified group, then that radix is returned. If the support package doesn't define a radix, or if no support package is loaded, then a system default radix is provided based on group width.

Exceptions Thrown:

None.

2.18.1.13 ILAModule.GetSupportPackageGroupSymbolFile

Description:

This method returns the full path of the default symbol file that the loaded support package associates with a given group.

Declaration Syntax:

Visual Basic

```
Function GetSupportPackageGroupSymbolFile
      (group As IChannelGroup) As String
```

C#

```
string GetSupportPackageGroupSymbolFile (IChannelGroup group)
```

C++

```
String* GetSupportPackageGroupSymbolFile (IChannelGroup* group)
```

Arguments:

`group` – Specifies the LA channel group for which the default symbol file is requested.

Return Value:

If no support packaged is loaded or a default symbol file is not defined, then the return value is String.Empty. Otherwise, a string representation of the full path and file name is returned. This value can be used with the ITlaSystem.LoadSymbolFile method.

Exceptions Thrown:

None.

Remarks:

Even when a support package is loaded, it might not define a default symbol file for existing current group. Clients of this method should check for return values of String.Empty.

2.18.1.14 ILAModule.ExportChannelSetup

Description:

This method exports channel setup into a file. When exporting files, the method is overloaded so that channel setup can be exported with default options; or an object of type ChannelImportExportOptions can be passed to specify how channel setup should be exported.

This method will return an integer to tell the ending status. 0 represents successful with no warnings. 1 represents successful with warnings. -1 represents stopped with errors.

Declaration Syntax:

Visual Basic

```
Function ExportChannelSetup (fileName As String) As Int32

Function ExportChannelSetup (fileName As String, _
      options As ChannelImportExportOptions) As Int32
```

C#

```
Int32 ExportChannelSetup (string fileName)

Int32 ExportChannelSetup (string fileName,
      ChannelImportExportOptions options)
```

C++

```
Int32 ExportChannelSetup (String* fileName)

Int32 ExportChannelSetup (String* fileName,
      ChannelImportExportOptions options)
```

Arguments:

`fileName` – The full path of the exported file.

`options` – A `ChannelImportExportOptions` object that defines which parameters of channel setup should be exported.

Return Value:

Returns an integer to indicate the ending status. 0 represents successful with no warnings. 1 represents successful with warnings. -1 represents stopped with errors.

Exceptions Thrown:

ArgumentNullException :
Condition: The `fileName` argument is null.
Message: Null argument passed to ExportChannelSetup.

PathTooLongException :
Condition: The file path contained in `filename` is too long.

Message: The passed-in file name is too long.

FileLoadException :
Condition: File can’t be written.
Message: Exported file cannot be written.

Remarks:

Since many problems can occur during file operations, the ExportChannelSetup method should always be called within a try block so exceptions can be caught. The above list of possible exceptions cannot be guaranteed complete, so a catchall clause can be used in addition to catching specific exceptions of interest.

2.18.1.15 ILAModule.ImportChannelSetup

Description:

This method imports channel setup from a file. When importing files, the method is overloaded so that channel setup can be imported with default options; or an object of type ChannelImportExportOptions can be passed to specify how channel setup should be imported.

This method will return an integer to tell the ending status. 0 represents successful with no warnings. 1 represents successful with warnings. -1 represents stopped with errors.

Declaration Syntax:

Visual Basic

```
Function ImportChannelSetup (fileName As String) As Int32

Function ImportChannelSetup (fileName As String, _
      options As ChannelImportExportOptions) As Int32
```

C#

```
Int32 ImportChannelSetup (string fileName)

Int32 ImportChannelSetup (string fileName,
      ChannelImportExportOptions options)
```

C++

```
Int32 ImportChannelSetup (String* fileName)

Int32 ImportChannelSetup (String* fileName,
      ChannelImportExportOptions options)
```

Arguments:

`fileName` – The full path of the imported file.

`options` – A `ChannelImportExportOptions` object that defines which parameters of channel setup should be imported.

Return Value:

Returns an integer to indicate the ending status. 0 represents successful with no warnings. 1 represents successful with warnings. -1 represents stopped with errors.

Exceptions Thrown:

TlaFileOperationException :
Condition: The imported file is not valid. When this is thrown the Failure member will be set to `InvalidFile`.
Message: The symbol file does not have a valid or recognizable format.

ArgumentNullException :

Condition: The `fileName` argument is null.
Message: Null argument passed to ImportChannelSetup.

ArgumentException :
Condition: The `fileName` argument is not a valid filename.
Message: Invalid file name passed to ImportChannelSetup.

PathTooLongException :
Condition: The file path contained in `filename` is too long.
Message: The passed-in file name is too long.

FileNotFoundException :
Condition: Specified imported file couldn't be found.
Message: Imported file not found.

FileLoadException :
Condition: File was found, but can't be loaded.
Message: Imported file cannot be loaded.

Remarks:

Since many problems can occur during file operations, the ImportChannelSetup method should always be called within a try block so exceptions can be caught. The above list of possible exceptions cannot be guaranteed complete, so a catchall clause can be used in addition to catching specific exceptions of interest.

2.19 ILAAcquisitionData

Description:

This interface extends the IAcquisitionData interface in order to provide clocking mode and status bit usage that were in use at the time of data acquisition.

Namespace: Tektronix.LogicAnalyzer.TpiNet

Declaration Syntax:

Visual Basic

```
Interface ILAAcquisitionData Inherits IAcquisitionData
```

C#

```
interface ILAAcquisitionData : IAcquisitionData
```

C++

```
__gc __interface ILAAcquisitionData : IAcquisitionData
```

Remarks:

ILAAcquisitionData notifies clients of changes in available acquisition samples due to changes in the sample suppression by raising the IValidity.ValidityChanged event.

2.19.1 ILAAcquisitionData Properties

2.19.1.1 ILAAcquisitionData.ClockMode

Description:

The read-only ClockMode property represents the module clock mode at the time data was acquired.

Declaration Syntax:

Visual Basic

```
ReadOnly Property ClockMode As ClockModeValue
```

C#

```
ClockModeValue ClockMode { get; }
```

C++

```
ClockModeValue get_ClockMode ();
```

Arguments:

None.

Exceptions Thrown:

InvalidOperationException :
    Condition: Status requested when status Valid property is false.
    Message: Module status values are not valid at this time.

Remarks:

The IAcquisitionData.Valid property should be checked prior to accessing this property.

2.19.1.2 ILAAcquisitionData.StatusBitsMode

Description:

The read-only StatusBitsMode property indicates the status bits of the LA were utilized at the time of the acquisition.

Declaration Syntax:

Visual Basic

```
ReadOnly Property StatusBitsMode As StatusBitsModeValue
```

C#

```
StatusBitsModeValue StatusBitsMode { get; }
```

C++

```
StatusBitsModeValue get_StatusBitsMode ();
```

Arguments:

None.

Exceptions Thrown:

InvalidOperationException :
    Condition: Status requested when status Valid property is false.
    Message: Module status values are not valid at this time.

Remarks:

None.

2.20 IScopeAcquisitionData

Description:

This interface extends the IAcquisitionData interface in order to provide per channel offset and range voltages for the DSO or iView module that were in use at the time of data acquisition.

Namespace: Tektronix.LogicAnalyzer.TpiNet

Declaration Syntax:

Visual Basic

```
Interface IScopeAcquisitionData Inherits IAcquisitionData
```

C#

```
interface IscopeAcquisitionData : IAcquisitionData
```

C++

```
__gc __interface IscopeAcquisitionData : IAcquisitionData
```

2.20.1 IScopeAcquisitionData Methods

2.20.1.1 IScopeAcquisitionData.ChannelOffset

Description:

This method returns the channel offset in volts.

Declaration Syntax:

Visual Basic

```
Function ChannelOffset (channel As Integer) As Double
```

C#

```
double ChannelOffset (int channel)
```

C++

```
double ChannelOffset (int channel)
```

Arguments:

channel – the channel number starting at 1.

Return Value:

Returns the channel offset in volts at the time data was acquired.

Exceptions Thrown:

ArgumentOutOfRangeException :
Condition: Invalid channel number passed in.
Message: Invalid channel number.

InvalidOperationException :
Condition: Status requested when status Valid property is false.
Message: Module status values are not valid at this time.

2.20.1.2 IScopeAcquisitionData.ChannelRange

Description:

This method returns the channel range in volts.

Declaration Syntax:

Visual Basic

```
Function ChannelRange (channel As Integer) As Double
```

C#

```
double ChannelRange (int channel)
```

C++

```
double ChannelRange (int channel)
```

Arguments:

channel – the channel number starting at 1.

Return Value:

Returns the channel range in volts at the time data was acquired.

Exceptions Thrown:

ArgumentOutOfRangeException :
    Condition: Invalid channel number passed in.
    Message: Invalid channel number.

InvalidOperationException :
    Condition: Status requested when status Valid property is false.
    Message: Module status values are not valid at this time.

2.21 IModuleRunStatus

Description:

Module types that can be started and stopped by the TLA system implement this interface in order to provide acquisition run control status. The interface provides notification of changes to the run state and a method to get detailed acquisition status information. This interface is provided by native module types and IAcquisitionPlugIn types to support the system run control loop, the system status monitor, and TPI.NET clients needing run time status information.

Namespace: Tektronix.LogicAnalyzer.TpiNet

Declaration Syntax:

Visual Basic

```
Interface IModuleRunStatus
```

C#

```
interface IModuleRunStatus
```

C++

```
__gc __interface IModuleRunStatus
```

Remarks:

Even though virtual modules cannot participate in acquisitions, they should implement this interface in order to provide detailed post acquisition status values.

2.21.1 IModuleRunStatus Properties

2.21.1.1 IModuleRunStatus.IsEnabled

Description:

If a data source can participate in system acquisitions, then the value of this property indicates whether it will acquire data during the next acquisition. This property is equivalent to using the "On/Off" button supplied by most modules in the system window. To disable a data source so that it will not be part of the next acquistion, set this value to false.

Declaration Syntax:

Visual Basic

```
Property IsEnabled As Boolean
```

C#

```
bool IsEnabled { set; get; }
```

C++

```
bool get_IsEnabled ();
void set_IsEnabled (bool);
```

Arguments:

None

Exceptions Thrown:

NotSupportedException:
    Condition: An attempt is made to set this property to true on a reference memory.
    Message: The data source cannot acquire data.

Remarks:

Not all data sources can acquire new data. Reference data sources are an example. Any data source object that cannot acquire data must set its IsEnabled values to false, and it will throw an exception if a client tries to enable it.

2.21.1.2 IModuleRunStatus.RunState

Description:

Returns the current run state of a module. The value of this property is a member of the RunStateValue enumeration.

Declaration Syntax:

Visual Basic

```
ReadOnly Property RunState As RunStateValue
```

C#

```
RunStateValue RunState { get; }
```

C++

```
RunStateValue get_RunState ();
```

Arguments:

None.

Exceptions Thrown:

None.

Remarks:

The value of this property should be updated in a progression from Idle to Complete and then back to idle. Modules may skip some states if they are not meaningful or if not caught by hardware; but they must have Idle, Ready, and Complete states.

2.21.2 IModuleRunStatus Methods

2.21.2.1 IModuleRunStatus.GetDetailedStatus

Description:

This method returns detailed acquisition status values. The IDetailedStatus is a base interface and this call may return an object with additional status values. For example, an LA module returns an ILADetailedStatus interface.

Declaration Syntax:

Visual Basic

```
Function GetDetailedStatus () As IDetailedStatus
```

C#

```
IDetailedStatus GetDetailedStatus ()
```

C++

```
IDetailedStatus* GetDetailedStatus ()
```

Arguments:

None.

Return Value:

Returns an IDetailedStatus interface.

Exceptions Thrown:

None.

Remarks:

A recommended implementation detail is to latch all pertinent run time status values during this call. This will help insure that the values are self consistent with a particular stage of the acquisition cycle.

2.21.3 IModuleRunStatus Events

2.21.3.1 IModuleRunStatus.RunStateChanged

Description:

When a module changes from one run state to another, the RunStateChanged event is raised.

Declaration Syntax:

Visual Basic

```
Event RunStateChanged As RunStateChangedHandler
```

C#

```
event RunStateChangedHandler RunStateChanged;
```

C++

```
__event RunStateChangedHandler RunStateChanged;
```

Event Data:

The run state of the module is passed as a parameter.

Examples:

C# code:

```
//
// Test generation of an IModuleRunStatus.RunStateChanged event.
//
private bool TestRunStateChanged()
{
        try
        {
                // Register for RunChanged event.

                // Need to use the remoter "shim" for remote clients.
                runStateChangedEventRemoter = new RunStateChangedRemoter();
                runStateChangedEventRemoter.RemoteEventOccurred +=
                        new RunStateChangedHandler( HandleRunStateChanged );

                // Register remoter "shim" with the RunStateChanged event.
                laModule.RunStateChanged += new RunStateChangedHandler
            ( m_runStateChangedEventRemoter.OnRemoteEvent );

                // Do a run to get some timebase data.
                // The RunCompleted handler will display the timebase info.
                m_tlaSystem.RunControl.Run();
        }
        catch ( InvalidOperationException ioEx )
        {
                // handle exception …
                return false;
        }

        return true;
}

//
// Handle an IModuleRunStatus.RunStateChanged event.
// Get RunStateValue, IsEnabled status and IDetailedStatus.
//
private void HandleRunStateChanged( object sender, RunStateEventArgs args )
{
        try
        {
        RunStateValue runStateValue = m_laModule.RunState;

        bool enabled = m_laModule.IsEnabled;
        m_laModule.IsEnabled = false;           // test disabling IsEnabled status

        IDetailedStatus status = m_laModule.GetDetailedStatus();
    }
    catch( InvalidOperationException ioEx )
    {
        // handle exception …
    }

        //
        // Unregister the RunStateChanged event.
        //
        m_laModule.RunStateChanged -= new RunStateChangedHandler
        ( m_runStateChangedEventRemoter.OnRemoteEvent );

        m_runStateChangedEventRemoter.RemoteEventOccurred -=
                        new RunStateChangedHandler( HandleRunStateChanged );

        m_runStateChangedEventRemoter = null;
}
```

2.22 IDetailedStatus

Description:

This interface provides access to detailed acquisition status. Instances of objects that implement this interface may be obtained from the module object before, during, and after acquisitions. This information is similar to what is displayed in the TLA Status Monitor dialog.

Namespace: Tektronix.LogicAnalyzer.TpiNet

Declaration Syntax:

Visual Basic

```
Interface IDetailedStatus
      Inherits IValidity
      Inherits IDisposable
```

C#

```
interface IDetailedStatus : IValidity, IDisposable
```

C++

```
__gc __interface IDetailedStatus : public IValidity, IDisposable
```

Remarks:

An iView module provides detailed status using just this interface. The DSO module and LA module inherit from this interface and provide additional status values.

2.22.1 IDetailedStatus Properties

2.22.1.1 IDetailedStatus.Valid

Description:

The read-only Valid property indicates if values provided by the other properties and methods are currently valid. Values may be invalid prior to the first acquisition by the module or after calibration or diagnostics.

Declaration Syntax:

Visual Basic

```
ReadOnly Property Valid As Boolean
```

C#

```
bool Valid { get; }
```

C++

```
bool get_Valid ();
```

Arguments:

None.

Exceptions Thrown:

None.

Remarks:

None.

Examples:

C# code:

```
//
// Get IDetailedStatus status.
//
public void TestIDetailedStatus( ILAModule laModule )
{
    IDetailedStatus status = laModule.GetDetailedStatus();

    // Get IDetailedStatus properties for the Main timebase.
    bool isValid = status.IsValid;
    DataSetValue dset = DataSetValue.Main;
    bool timeBaseSupported = status.TimebaseStatusSupported( dset );
    RunStateValue runStateVal = status.TimebaseRunState( dset );
    long sampleDepth = status.TimebaseSampleDepth( dset );
    int tbtPercent = status.TimebaseTriggerPercent( dset );
    long tbtPostfill = status.TimebaseTriggerPostfill( dset );
    long tbtPrefill = status.TimebaseTriggerPrefill( dset );

    // Use the properties just obtained …
}
```

2.22.2 IDetailedStatus Methods

2.22.2.1 IDetailedStatus.TimebaseStatusSupported

Description:

This method returns an indication of if status reporting for the timebase is supported.

Declaration Syntax:

Visual Basic

```
Function TimebaseStatusSupported ( dset As DataSetValue ) As Boolean
```

C#

```
bool TimebaseStatusSupported (DataSetValue dset)
```

C++

```
bool TimebaseStatusSupported (DataSetValue dset)
```

Arguments:

dset – the timebase of interest.

Return Value:

Returns true if status reporting is supported for that timebase.

Exceptions Thrown:

InvalidOperationException :
Condition: Status requested when status Valid property is false.
Message: Module detailed status values are not valid at this time.

Remarks:

Use this method to make sure the indicated timebase is supported before calling the other TimebaseXXX methods. If not supported, they will throw exceptions.

LA supports timebase reporting for Main and MagniVu only. DSO and iView support timebase reporting for Main only.

Examples:

See IDetailedStatus.Valid.

2.22.2.2 IDetailedStatus.TimebaseRunState

Description:

This method returns the acquisition run control state for the given timebase. A virtual module has a fixed run state of Idle.

Declaration Syntax:

Visual Basic

```
Function TimebaseRunState ( dset As DataSetValue ) As RunStateValue
```

C#

```
RunStateValue TimebaseRunState (DataSetValue dset)
```

C++

```
RunStateValue TimebaseRunState (DataSetValue dset)
```

Arguments:

dset – the timebase of interest.

Return Value:

Returns the acquisition run control state for the timebase.

Exceptions Thrown:

InvalidOperationException :
Condition: Status requested when status Valid property is false.
Message: Module detailed status values are not valid at this time.

ArgumentException :
Condition: Status for the given data set is not supported.
Message: Status for the given data set is not supported.

Remarks:

Use TimbaseStatusSupported() for the data to determine if status from the data set is supported.

Examples:

See IDetailedStatus.Valid.

2.22.2.3 IDetailedStatus.TimebaseSampleDepth

Description:

This method returns the memory depth in samples for the given timebase.

Declaration Syntax:

Visual Basic

```
Function TimebaseSampleDepth ( dset As DataSetValue ) As Long
```

C#

```
long TimebaseSampleDepth (DataSetValue dset)
```

C++

```
__int64 TimebaseSampleDepth (DataSetValue dset)
```

Arguments:

dset – the timebase of interest.

Return Value:

Returns the memory depth in samples.

Exceptions Thrown:

InvalidOperationException :
Condition: Status requested when status Valid property is false.
Message: Module detailed status values are not valid at this time.

ArgumentException :
Condition: Status for the given data set is not supported.
Message: Status for the given data set is not supported.

Remarks:

Use TimbaseStatusSupported() for the data to determine if status from the data set is supported.

Examples:

See IDetailedStatus.Valid.

2.22.2.4 IDetailedStatus.TimebaseTriggerPercent

Description:

This method returns the trigger position as a percent from 1 to 100 for the given timebase.

Description:

This method returns the trigger position as a percent from 1 to 100 for the given timebase.

Declaration Syntax:

Visual Basic

```
Function TimebaseTriggerPercent ( dset As DataSetValue ) As Integer
```

C#

```
int TimebaseTriggerPercent (DataSetValue dset)
```

C++

```
int TimebaseTriggerPercent (DataSetValue dset)
```

Arguments:

dset – the timebase of interest.

Return Value:

Returns the trigger position as a percent from 1 to 100.

Exceptions Thrown:

InvalidOperationException :
Condition: Status requested when status Valid property is false.
Message: Module detailed status values are not valid at this time.

ArgumentException :
Condition: Status for the given data set is not supported.
Message: Status for the given data set is not supported.

Remarks:

Use TimbaseStatusSupported() for the data to determine if status from the data set is supported.

Examples:

See IDetailedStatus.Valid.

2.22.2.5 IDetailedStatus.TimebaseTriggerPrefill

Description:

This method returns the number of pre-trigger samples filled for the given timebase.

Declaration Syntax:

Visual Basic

```
Function TimebaseTriggerPrefill ( dset As DataSetValue ) As Long
```

C#

```
long TimebaseTriggerPrefill (DataSetValue dset)
```

C++

```
__int64 TimebaseTriggerPrefill (DataSetValue dset)
```

Arguments:

dset – the timebase of interest.

Return Value:

Returns the number of pre-trigger samples filled.

Exceptions Thrown:

InvalidOperationException :
Condition: Status requested when status Valid property is false.
Message: Module detailed status values are not valid at this time.

ArgumentException :
Condition: Status for the given data set is not supported.
Message: Status for the given data set is not supported.

Remarks:

Use TimbaseStatusSupported() for the data to determine if status from the data set is supported.

Examples:

See IDetailedStatus.Valid.

2.22.2.6 IDetailedStatus.TimebaseTriggerPostfill

Description:

This method returns the number of post-trigger samples filled for the given timebase.

Declaration Syntax:

Visual Basic

```
Function TimebaseTriggerPostfill ( dset As DataSetValue ) As Long
```

C#

```
long TimebaseTriggerPostfill (DataSetValue dset)
```

C++

```
__int64 TimebaseTriggerPostfill (DataSetValue dset)
```

Arguments:

dset – the timebase of interest.

Return Value:

Returns the number of post-trigger samples filled.

Exceptions Thrown:

InvalidOperationException :
Condition: Status requested when status Valid property is false.
Message: Module detailed status values are not valid at this time.

ArgumentException :
Condition: Status for the given data set is not supported.
Message: Status for the given data set is not supported.

Remarks:

Use TimbaseStatusSupported() for the data to determine if status from the data set is supported.

Examples:

See IDetailedStatus.Valid.

2.23 IDSODetailedStatus

Description:

This interface provides access to detailed DSO acquisition status.

Namespace: Tektronix.LogicAnalyzer.TpiNet

Declaration Syntax:

Visual Basic

```
Interface IDSODetailedStatus Inherits IDetailedStatus
```

C#

```
interface IDSODetailedStatus : IDetailedStatus
```

C++

```
__gc __interface IDetailedStatus : public IDetailedStatus
```

Remarks:

This interface simply adds a SigArmOutAsserted indicator to the IDetailedStatus interface.

2.23.1 IDSODetailedStatus Properties

2.23.1.1 IDSODetailedStatus.SigArmOutAsserted

Description:

The read-only SigArmOutAsserted property provides the state of the Signal/Arm Out line from the module.

Declaration Syntax:

Visual Basic

```
ReadOnly Property SigArmOutAsserted As Boolean
```

C#

```
bool SigArmOutAsserted { get; }
```

C++

```
bool get_SigArmOutAsserted ();
```

Arguments:

None.

Exceptions Thrown:

InvalidOperationException :
Condition: Status requested when status Valid property is false.
Message: Module detailed status values are not valid at this time.

Remarks:

None.

Examples:

C# code:

```
//
// Get the base class, IDetailedStatus, status.
//
public void TestIDetailedStatus( ILAModule laModule )
{
    IDetailedStatus status = laModule.GetDetailedStatus();

    // Get IDetailedStatus properties for the Main timebase.
    bool isValid = status.IsValid;
    DataSetValue dset = DataSetValue.Main;
    bool timeBaseSupported = status.TimebaseStatusSupported( dset );
    RunStateValue runStateVal = status.TimebaseRunState( dset );
    long sampleDepth = status.TimebaseSampleDepth( dset );
    int tbtPercent = status.TimebaseTriggerPercent( dset );
    long tbtPostfill = status.TimebaseTriggerPostfill( dset );
    long tbtPrefill = status.TimebaseTriggerPrefill( dset );

    // Use the properties just obtained …
}

//
// IDSODetailedStatus example.
//
public void TestIDSODetailedStatus( ILAModule laModule )
{
        ILADetailedStatus status = (ILADetailedStatus) laModule.GetDetailedStatus();

        //
        // Test the base class properties.
        //
    TestIDetailedStatus( status );

        //
        // Test the derived class properties.
        //
        bool sigArmOutAsserted = status.SigArmOutAsserted;
}
```

2.24 ILADetailedStatus

Description:

This interface provides access to detailed LA acquisition status.

Namespace: Tektronix.LogicAnalyzer.TpiNet

Declaration Syntax:

Visual Basic

```
Interface ILADetailedStatus Inherits IDetailedStatus
```

C#

```
interface ILADetailedStatus : IDetailedStatus
```

C++

```
__gc __interface ILADetailedStatus : public IDetailedStatus
```

Remarks:

This interface extends the IDetailedStatus interface with LA specific status values.

2.24.1 ILADetailedStatus Properties

2.24.1.1 ILADetailedStatus.SigArmOutAsserted

Description:

The read-only SigArmOutAsserted property provides the state of the Signal/Arm Out line from the module.

Declaration Syntax:

Visual Basic

```
ReadOnly Property SigArmOutAsserted As Boolean
```

C#

```
bool SigArmOutAsserted { get; }
```

C++

```
bool get_SigArmOutAsserted ();
```

Arguments:

None.

Exceptions Thrown:

InvalidOperationException :
Condition: Status requested when status Valid property is false.
Message: Module detailed status values are not valid at this time.

Remarks:

None.

Examples:

C# code:

```
//
// Get the base class, IDetailedStatus, status.
//
public void TestIDetailedStatus( ILAModule laModule )
{
    IDetailedStatus status = laModule.GetDetailedStatus();

    // Get IDetailedStatus properties for the Main timebase.
    bool isValid = status.IsValid;
    DataSetValue dset = DataSetValue.Main;
    bool timeBaseSupported = status.TimebaseStatusSupported( dset );
    RunStateValue runStateVal = status.TimebaseRunState( dset );
    long sampleDepth = status.TimebaseSampleDepth( dset );
```

```
      int tbtPercent = status.TimebaseTriggerPercent( dset );
      long tbtPostfill = status.TimebaseTriggerPostfill( dset );
      long tbtPrefill = status.TimebaseTriggerPrefill( dset );

      // Use the properties just obtained …
}

//
// ILADetailedStatus example.
//
public void TestILADetailedStatus( ILAModule laModule )
{
            ILADetailedStatus status = (ILADetailedStatus) laModule.GetDetailedStatus();

            //
            // Test the base class properties.
            //
      TestIDetailedStatus( status );

            //
            // Test the derived class properties.
            //
            bool sigArmOutAsserted = status.SigArmOutAsserted;
            bool sigInAsserted = status.SigInAsserted;
            int  triggerState  = status.TriggerState;
            bool clockActive   = status.ClockActive;
            CompareStatusValue compareStatus = status.CompareStatus;

            // Allocate each counterTimer resource.
            int counterResource = 1;
            int timerResource = 2;

            int maxCountersTimers = status.MaxCountersTimers();
            CounterTimerModeValue counterTimerMode =
            status.CounterTimerMode(counterResource);
            bool counterTimerHasTarget =
            status.CounterTimerHasTarget(counterResource);
            long counterValue = status.CounterValue(counterResource);
            long counterTarget = status.CounterTarget(counterResource);
            Decimal timerValue = status.TimerValue(timerResource);
            Decimal timerTarget = status.TimerTarget(timerResource);
}
```

2.24.1.2 ILADetailedStatus.SigInAsserted

Description:

The read-only SigInAsserted property provides the state of the Signal Input line from the module.

Declaration Syntax:

Visual Basic

```
ReadOnly Property SigInAsserted As Boolean
```

C#

```
bool SigInAsserted { get; }
```

C++

```
bool get_SigInAsserted ();
```

Arguments:

None.

Exceptions Thrown:

InvalidOperationException :
Condition: Status requested when status Valid property is false.
Message: Module detailed status values are not valid at this time.

Remarks:

None.

Examples:

See ILADetailedStatus.SigArmOutAsserted.

2.24.1.3 ILADetailedStatus.TriggerState

Description:

The read-only TriggerState property provides the trigger machine state from the module.

Declaration Syntax:

Visual Basic

```
ReadOnly Property TriggerState As Integer
```

C#

```
int TriggerState { get; }
```

C++

```
int get_TriggerState ();
```

Arguments:

None.

Exceptions Thrown:

InvalidOperationException :
Condition: Status requested when status Valid property is false.
Message: Module detailed status values are not valid at this time.

Remarks:

The trigger state numbering starts at 1.

Examples:

See ILADetailedStatus.SigArmOutAsserted.

2.24.1.4 ILADetailedStatus.ClockActive

Description:

The read-only ClockActive property provides an indication of functioning clock signal.

Declaration Syntax:

Visual Basic

```
ReadOnly Property ClockActive As Boolean
```

C#

```
bool ClockActive { get; }
```

C++

```
bool get_ClockActive ();
```

Arguments:

None.

Exceptions Thrown:

InvalidOperationException :
Condition: Status requested when status Valid property is false.
Message: Module detailed status values are not valid at this time.

Remarks:

None.

Examples:

See ILADetailedStatus.SigArmOutAsserted.

2.24.1.5 ILADetailedStatus.CompareStatus

Description:

The read-only CompareStatus property provides the status of repetitive compare on the module.

Declaration Syntax:

Visual Basic

```
ReadOnly Property CompareStatus As CompareStatusValue
```

C#

```
CompareStatusValue CompareStatus { get; }
```

C++

```
CompareStatusValue get_CompareStatus ();
```

Arguments:

None.

Exceptions Thrown:

InvalidOperationException :
Condition: Status requested when status Valid property is false.
Message: Module detailed status values are not valid at this time.

Remarks:

None.

Examples:

See ILADetailedStatus.SigArmOutAsserted.

2.24.2 ILADetailedStatus Methods

2.24.2.1 ILADetailedStatus.MaxCountersTimers

Description:

This method returns the number of counter/timer resources supported by the module.

Declaration Syntax:

Visual Basic

```
Function MaxCountersTimers () As Integer
```

C#

```
int MaxCountersTimers ()
```

C++

```
int MaxCountersTimers ()
```

Arguments:

None.

Return Value:

Returns the number of counter/timer resources available for the module. LA modules supply two counter/timer resources.

Exceptions Thrown:

InvalidOperationException :
Condition: Status requested when status Valid property is false.
Message: Module detailed status values are not valid at this time.

Remarks:

Use this method to limit the index used for the other counter/timer methods. Indices are zero based, so MaxCountersTimers-1 is upper limit.

Examples:

See ILADetailedStatus.SigArmOutAsserted.

2.24.2.2 ILADetailedStatus.CounterTimerMode

Description:

This method returns counter/timer resource utilization.

Declaration Syntax:

Visual Basic

```
Function CounterTimerMode (index As Integer) As CounterTimerModeValue
```

C#

```
CounterTimerModeValue CounterTimerMode (int index)
```

C++

```
CounterTimerModeValue CounterTimerMode (int index)
```

Arguments:

index – which counter/timer resource mode to request

Return Value:

Returns the mode that indicates Unused, Counter, or Timer.

Exceptions Thrown:

InvalidOperationException :
Condition: Status requested when status Valid property is false.
Message: Module detailed status values are not valid at this time.

ArgumentOutOfRangeException :
Condition: Invalid index is passed in.
Message: Counter/Timer index out of the range 1 to 2.

Remarks:

Use this method to determine how the module uses the counter/timer resource.

Examples:

See ILADetailedStatus.SigArmOutAsserted.

2.24.2.3 ILADetailedStatus.CounterTimerHasTarget

Description:

This method is used to determine if the counter/timer is has a target value associated with being used as an event in the trigger program.

Declaration Syntax:

Visual Basic

```
Function CounterTimerHasTarget (index As Integer) As Boolean
```

C#

```
bool CounterTimerHasTarget (int index)
```

C++

```
bool CounterTimerHasTarget (int index)
```

Arguments:

index – which counter/timer to check

Return Value:

Returns true if the counter/timer has a target value.

Exceptions Thrown:

InvalidOperationException :
   Condition: Status requested when status Valid property is false.
   Message: Module detailed status values are not valid at this time.

ArgumentOutOfRangeException :
   Condition: Invalid index is passed in.
   Message: Counter/Timer index out of the range 1 to 2.

Remarks:

Use this method to before calling ILADetailedStatus.CounterTarget() and ILADetailedStatus.TimerTarget().

Examples:

See ILADetailedStatus.SigArmOutAsserted.

2.24.2.4 ILADetailedStatus.CounterValue

Description:

This method returns the value of the trigger program counter.

Declaration Syntax:

Visual Basic

```
Function CounterValue (index As Integer) As Long
```

C#

```
long CounterValue (int index)
```

C++

```
__int64 CounterValue (int index)
```

Arguments:

index – which counter value to return.

Return Value:

Returns the value of the counter.

Exceptions Thrown:

InvalidOperationException :
Condition: Status requested when status Valid property is false.
Message: Module detailed status values are not valid at this time.

InvalidOperationException :
Condition: Resource identified by index is not a counter.
Message: Requested trigger counter does not exist.

ArgumentOutOfRangeException :
Condition: Invalid index is passed in.
Message: Counter/Timer index out of the range 1 to 2.

Remarks:

Use ILADetailedStatus.CounterTimerMode() before this method to insure it is a counter.

Examples:

See ILADetailedStatus.SigArmOutAsserted.

2.24.2.5 ILADetailedStatus.CounterTarget

Description:

This method returns the target value of the trigger program counter.

Declaration Syntax:

Visual Basic

```
Function CounterTarget (index As Integer) As Long
```

C#

```
long CounterTarget (int index)
```

C++

```
__int64 CounterTarget (int index)
```

Arguments:

index – which counter target value to return.

Return Value:

Returns the target value of the counter.

Exceptions Thrown:

InvalidOperationException :
Condition: Status requested when status Valid property is false.
Message: Module detailed status values are not valid at this time.

InvalidOperationException :
Condition: Resource identified by index is not a counter.
Message: Requested trigger counter does not exist.

InvalidOperationException :
Condition: Requested counter does not have a target value.
Message: Requested trigger counter does not have a target value.

ArgumentOutOfRangeException :
Condition: Invalid index is passed in.
Message: Counter/Timer index out of the range 1 to 2.

Remarks:

Use ILADetailedStatus.CounterTimerMode() before this method to insure it is a counter. Use ILADetailedStatus.CounterTimerHasTarget() before this method to insure it has a target value.

Examples:

See ILADetailedStatus.SigArmOutAsserted.

2.24.2.6 ILADetailedStatus.TimerValue

Description:

This method returns the value of the trigger program timer in picoseconds.

Declaration Syntax:

Visual Basic

```
Function TimerValue (index As Integer) As Decimal
```

C#

```
Decimal TimerValue (int index)
```

C++

```
Decimal TimerValue (int index)
```

Arguments:

index – which timer value to return.

Return Value:

Returns the value of the timer.

Exceptions Thrown:

InvalidOperationException :
  Condition: Status requested when status Valid property is false.
  Message: Module detailed status values are not valid at this time.

InvalidOperationException :
  Condition: Resource identified by index is not a timer.
  Message: Requested trigger timer does not exist.

ArgumentOutOfRangeException :
  Condition: Invalid index is passed in.
  Message: Counter/Timer index out of the range 1 to 2.

Remarks:

Use ILADetailedStatus.CounterTimerMode() before this method to insure it is a timer.

Examples:

See ILADetailedStatus.SigArmOutAsserted.

2.24.2.7 ILADetailedStatus.TimerTarget

Description:

This method returns the target value of the trigger program timer in picoseconds.

Declaration Syntax:

Visual Basic

```
Function TimerTarget (index As Integer) As Decimal
```

C#

```
Decimal TimerTarget (int index)
```

C++

```
Decimal TimerTarget (int index)
```

Arguments:

index – which timer target value to return.

Return Value:

Returns the target value of the timer.

Exceptions Thrown:

InvalidOperationException :
    Condition: Status requested when status Valid property is false.
    Message: Module detailed status values are not valid at this time.

InvalidOperationException :
    Condition: Resource identified by index is not a timer.
    Message: Requested trigger timer does not exist.

InvalidOperationException :
    Condition: Requested timer does not have a target value.
    Message: Requested trigger timer does not have a target value.

ArgumentOutOfRangeException :
    Condition: Invalid index is passed in.
    Message: Counter/Timer index out of the range 1 to 2.

Remarks:

Use ILADetailedStatus.CounterTimerMode() before this method to insure it is a timer. Use ILADetailedStatus.CounterTimerHasTarget() before this method to insure it has a target value.

Examples:

See ILADetailedStatus.SigArmOutAsserted.

2.25 IFilters

Description:

This interface provides access to the list of filters defined for an LA.

Namespace: Tektronix.LogicAnalyzer.TpiNet

Declaration Syntax:

Visual Basic

```
Interface IFilters
```

C#

```
interface IFilters
```

C++

```
__gc __interface IFilters
```

2.25.1 IFilters Properties

2.25.1.1 IFilters.List

Description:

Gets a list of all filters defined for the instrument.

Declaration Syntax:

Visual Basic

```
ReadOnly Property List As ArrayList
```

C#

```
ArrayList List { get;}
```

C++

```
ArrayList* get_List ();
```

Arguments:

None.

Exceptions Thrown:

InvalidOperationException :
    Condition: Attempt is made to change the list.
    Message: List of filters is read-only.

Remarks:

The value of this property is a read-only ArrayList object. Although getting the property value never throws an exception, any attempt to modify the list will throw an exception. This means that the number of available filters in the instrument cannot be changed by operations on this ArrayList.

2.25.2 IFilters Events

2.25.2.1 IFilters.ListChanged

Description:

When filters are added or removed from the instrument, the IFilters raises this event. Clients that utilize filters should obtain a fresh list of filters from the IFilters.List property when this event is received.

Declaration Syntax:

Visual Basic

```
Event ListChanged As ObjectEventHandler
```

C#

```
event ObjectEventHandler ListChanged;
```

C++

```
__event ObjectEventHandler ListChanged;
```

Event Data:

The IFilters object is passed as a parameter.

2.26 IFilter

Description:

This interface provides access to a filter defined for an LA.

Namespace: Tektronix.LogicAnalyzer.TpiNet

Declaration Syntax:

Visual Basic

```
Interface IFilter
```

C#

```
interface IFilter
```

C++

```
__gc __interface IFilter
```

Remarks:

Filters may be defined which use one or more disassembly groups. Since TPI clients don't have a disassembler running, the output of such a filter will be different than for a Listing Window

2.26.1 IFilter Properties

2.26.1.1 IFilter.Name

Description:

The Name property provides the filter with a modifiable name.

Declaration Syntax:

Visual Basic

```
Property Name As String
```

C#

```
string Name { get; set;}
```

C++

```
String* get_Name ();
void set_Name ( String* name );
```

Arguments:

name – a new name for the filter supplied to the setter.

Exceptions Thrown:

InvalidOperationException :
Condition: Attempt is made to modify filter on a read only setup.
Message: Filter setup is read-only.

Remarks:

Changing the filter name raises the NameChanged event of the filter.

Clients should check the IDataSource.IsReadOnlySetup property before renaming a filter.

2.26.1.2 IFilter.Description

Description:

The Description property provides the filter with a modifiable description.

Declaration Syntax:

Visual Basic

```
Property Description As String
```

C#

```
string Description { get; set;}
```

C++

```
String* get_Description ();
void set_Description ( String* name );
```

Arguments:

description – a new description for the filter supplied to the setter.

Exceptions Thrown:

InvalidOperationException :
Condition: Attempt is made to modify filter on a read only setup.
Message: Filter setup is read-only.

Remarks:

Changing the filter description raises the DescriptionChanged event of the filter.

Clients should check the IDataSource.IsReadOnlySetup property before modifying a filter.

2.26.2 IFilter Methods

2.26.2.1 IFilter.ContainsShowOrHide

Description:

This method indicates if the filter has any show/hide actions for sample records.

Declaration Syntax:

Visual Basic

```
Function ContainsShowOrHide () As Boolean
```

C#

```
bool ContainsShowOrHide ()
```

C++

```
bool ContainsShowOrHide ()
```

Arguments:

None.

Return Value:

Returns true if the filter includes show or hide actions. Returns false if the filter does not include any show or hide actions.

Exceptions Thrown:

None.

2.26.2.2 IFilter.ShownAtOrAfter

Description:

This method returns the next shown sample at or after the fromSample.

Declaration Syntax:

Visual Basic

```
Function ShownAtOrAfter ( fromSample As Long, maxSamples As Long ) As
Long
```

C#

```
long ShownAtOrAfter ( long fromSample, long maxSamples )
```

C++

```
__int64 ShownAtOrAfter ( __int64 fromSample, __int64 maxSamples )
```

Arguments:

fromSample – beginning sample number to start looking for a shown sample.
maxSamples – the maximum number of samples to search before returning not found.

Return Value:

Returns a sample number greater than or equal to zero when a shown sample can be located. Returns –1 to indicate not found. A sample may not be found when no shown samples have been located within the maxSamples limit or when no shown samples have been located and the final acquisition sample has been checked.

Exceptions Thrown:

None.

2.26.2.3 IFilter.ShownAtOrBefore

Description:

This method returns the next shown sample at or before the fromSample.

Declaration Syntax:

Visual Basic

```
Function ShownAtOrBefore ( fromSample As Long, maxSamples As Long ) As
Long
```

C#

```
long ShownAtOrBefore ( long fromSample, long maxSamples )
```

C++

```
__int64 ShownAtOrBefore ( __int64 fromSample, __int64 maxSamples )
```

Arguments:

fromSample – beginning sample number to start looking for a shown sample.
maxSamples – the maximum number of samples to search before returning not found.

Return Value:

Returns a sample number greater than or equal to zero when a shown sample can be located. Returns –1 to indicate not found. A sample may not be found when no shown samples have been located within the maxSamples limit or when no shown samples have been located and the first acquisition sample has been checked.

Exceptions Thrown:

None.

2.26.2.4 IFilter.ContainsColor

Description:

This method indicates if the filter has any color actions for the designated group or any color actions that apply to all groups.

Declaration Syntax:

Visual Basic

```
Function ContainsColor ( groupName As String ) As Boolean
```

C#

```
bool ContainsColor ( string groupName )
```

C++

```
bool ContainsColor ( String* groupName )
```

Arguments:

groupName – the name of a group that the filter might provide colors for.

Return Value:

Returns true if the filter includes any color actions for the designated group or any color actions that apply to all groups.

Exceptions Thrown:

ArgumentException :
Condition: No group with specified name exists.
Message: Specified group does not exist.

2.26.2.5 IFilter.ColorAtOrAfter

Description:

This method returns the next sample at or after the fromSample that the filter provides color to for the specified group.

Declaration Syntax:

Visual Basic

```
Function ColorAtOrAfter ( fromSample As Long, maxSamples As Long,
groupName As String ) As Long
```

C#

```
long ColorAtOrAfter ( long fromSample, long maxSamples, string
groupName )
```

C++

```
__int64 ColorAtOrAfter ( __int64 fromSample, __int64 maxSamples,
String* groupName )
```

Arguments:

fromSample – beginning sample number to start looking for a shown sample.
maxSamples – the maximum number of samples to search before returning not found.
groupName – the group to search for a color sample for.

Return Value:

Returns a sample number greater than or equal to zero when a colored sample can be located. Returns –1 to indicate not found. A sample may not be found when no colored samples have been located within the maxSamples limit or when no colored samples have been located and the final acquisition sample has been checked.

Exceptions Thrown:

None.

2.26.2.6 IFilter.ColorAtOrBefore

Description:

This method returns the next sample at or before the fromSample that the filter provides color to for the specified group.

Declaration Syntax:

Visual Basic

```
Function ColorAtOrBefore ( fromSample As Long, maxSamples As Long,
groupName As String ) As Long
```

C#

```
long ColorAtOrBefore ( long fromSample, long maxSamples, string
groupName )
```

C++

```
__int64 ColorAtOrBefore ( __int64 fromSample, __int64 maxSamples,
String* groupName )
```

Arguments:

fromSample – beginning sample number to start looking for a shown sample.
maxSamples – the maximum number of samples to search before returning not found.
groupName – the group to search for a color sample for.

Return Value:

Returns a sample number greater than or equal to zero when a colored sample can be located. Returns –1 to indicate not found. A sample may not be found when no colored samples have been located within the maxSamples limit or when no colored samples have been located and the first acquisition sample has been checked.

Exceptions Thrown:

None.

2.26.2.7 IFilter.ColorAt

Description:

This method is used to retrieve the colors defined by the filter for the group on the sample number.

Declaration Syntax:

Visual Basic

```
Function ColorAt ( sample As Long, groupName As String, foreground As
Color, background As Color ) As Boolean
```

C#

```
bool ColorAt ( long sample, string groupName, out Color foreground, out
Color background )
```

C++

```
bool ColorAt ( __int64 sample, String* groupName, Color* foreground,
Color* background )
```

Arguments:

sample –sample number to get color information for.
groupName – the group to get color information for.
foreground – a reference to the foreground color is returned in this parameter.
background – a reference to the background color is returned in this parameter.

Return Value:

Returns a true when the filter defines color values for the group at the specified sample. On success, the foreground and background colors are returned through output parameters. Returns false when the filter has no color definitions for the group at the specified sample. On failure, the foreground and background parameters are unchanged.

Exceptions Thrown:

ArgumentException :
    Condition: No group with specified name exists.
    Message: Specified group does not exist.

2.26.3 IFilter Events

2.26.3.1 IFilter.NameChanged

Description:

When the name of a filter is changed, the NameChanged event is raised.

Declaration Syntax:

Visual Basic

```
Event NameChanged As ObjectEventHandler
```

C#

```
event ObjectEventHandler NameChanged;
```

C++

```
__event ObjectEventHandler NameChanged;
```

Event Data:

The IFilter object is passed as a parameter.

2.26.3.2 IFilter.DescriptionChanged

Description:

When the description of a filter is changed, the DescriptionChanged event is raised.

Declaration Syntax:

Visual Basic

```
Event DescriptionChanged As ObjectEventHandler
```

C#

```
event ObjectEventHandler DescriptionChanged;
```

C++

```
__event ObjectEventHandler DescriptionChanged;
```

Event Data:

The IFilter object is passed as a parameter.

2.26.3.3 IFilter.DefinitionChanged

Description:

When the definition of a filter is changed, the DefinitionChanged event is raised. The filter definition refers to complete body of the filter consisting of filter clauses, their conditions, and actions.

Declaration Syntax:

Visual Basic

```
Event DefinitionChanged As ObjectEventHandler
```

C#

```
event ObjectEventHandler DefinitionChanged;
```

C++

```
__event ObjectEventHandler DefinitionChanged;
```

Event Data:

The IFilter object is passed as a parameter.

2.27 IThresholds

Description:

This interface provides access to the LA probe thresholds.

Namespace: Tektronix.LogicAnalyzer.TpiNet

Declaration Syntax:

Visual Basic

```
Interface IThresholds
```

C#

```
interface IThresholds
```

C++

```
__gc __interface IThresholds
```

2.27.1 IThresholds Properties

2.27.1.1 IThresholds.Length

Description:

The read-only Length property provides the number of settable threshold items contained by the module.

Declaration Syntax:

Visual Basic

```
ReadOnly Property Length As Integer
```

C#

```
int Length { get; }
```

C++

```
int get_Length ();
```

Arguments:

None.

Exceptions Thrown:

None.

2.27.1.2 IThresholds.DefaultVolts

Description:

The read only DefaultVolts property provides the default threshold value in Volts for the module.

Declaration Syntax:

Visual Basic

```
ReadOnly Property DefaultVolts As Double
```

C#

```
double DefaultVolts () { get; }
```

C++

```
double get_DefaultVolts ();
```

Arguments:

None.

Exceptions Thrown:

None.

Remarks:

This returns zero if the instrument containing the IThresholds object has been disposed.

2.27.1.3 IThresholds.MinimumVolts

Description:

The read only MinimumVolts property provides the minimum allowable threshold value in Volts for the module.

Declaration Syntax:

Visual Basic

```
ReadOnly Property MinimumVolts As Double
```

C#

```
double MinimumVolts () { get; }
```

C++

```
double get_MinimumVolts ();
```

Arguments:

None.

Exceptions Thrown:

This returns zero if the instrument containing the IThresholds object has been disposed.

2.27.1.4 IThresholds.MaximumVolts

Description:

The read only MaximumVolts property provides the maximum allowable threshold value in Volts for the module.

Declaration Syntax:

Visual Basic

```
ReadOnly Property MaximumVolts As Double
```

C#

```
double MaximumVolts () { get; }
```

C++

```
double get_MaximumVolts ();
```

Arguments:

None.

Exceptions Thrown:

None.

Remarks:

This returns zero if the instrument containing the IThresholds object has been disposed.

2.27.1.5 IThresholds.StepVolts

Description:

The read only StepVolts property provides the minimum allowable threshold step value in Volts for the module.

Declaration Syntax:

Visual Basic

```
ReadOnly Property StepVolts As Double
```

C#

```
double StepVolts () { get; }
```

C++

```
double get_StepVolts ();
```

Arguments:

None.

Exceptions Thrown:

None.

Remarks:

This returns zero if the instrument containing the IThresholds object has been disposed.

2.27.2 IThresholds Methods

2.27.2.1 IThresholds.GetName

Description:

This method provides the name of the channel or channel sections of a threshold item.

Declaration Syntax:

Visual Basic

```
Function GetName ( index As Integer ) As String
```

C#

```
string GetName ( int index )
```

C++

```
String GetName ( int index )
```

Arguments:

Index – the threshold item index in the range zero through Length - 1.

Return Value:

Returns the name of the channel or channel sections of the threshold item.

Exceptions Thrown:

ArgumentOutOfRangeException :
Condition: An invalid index is supplied.

Remarks:

If the instrument containing the IThresholds object has been disposed, the index is considered invalid and an ArgumentOutOfRangeException is thrown.

2.27.2.2 IThresholds.GetVolts

Description:

This method provides the threshold value in Volts for the channel or section of channels represented by the indexed threshold item.

Declaration Syntax:

Visual Basic

```
Function GetVolts ( index As Integer ) As Double
```

C#

```
double GetVolts ( int index )
```

C++

```
double GetVolts ( int index )
```

Arguments:

Index – the threshold item index in the range zero through Length - 1.

Return Value:

Returns the threshold value in volts.

Exceptions Thrown:

ArgumentOutOfRangeException :
    Condition: An invalid index is supplied.

Remarks:

If the instrument containing the IThresholds object has been disposed, the index is considered invalid and an ArgumentOutOfRangeException is thrown.

2.27.2.3 IThresholds.SetVolts

Description:

This method sets the threshold value in Volts for the channel or section of channels represented by the indexed threshold item. This method will coerce the volts value to the nearest legal threshold for the module if the input threshold is out of bounds or not at a legal step value.

Declaration Syntax:

Visual Basic

```
Subroutine SetVolts ( index As Integer, Double As volts )
```

C#

```
void SetVolts ( int index, double volts )
```

C++

```
void SetVolts ( int index, double volts )
```

Arguments:

index – the threshold item index in the range zero through Length - 1.
volts – the threshold value in volts.

Return Value:

Returns the threshold value in volts.

Exceptions Thrown:

ArgumentOutOfRangeException :
    Condition: An invalid index is supplied.

InvalidOperationException :
    Condition: Attempt is made to modify threshold on a read only setup.
    Message: Instrument setup is read-only.

Remarks:

If the instrument containing the IThresholds object has been disposed, the index is considered invalid and an ArgumentOutOfRangeException is thrown.

2.28 ICompareSetup

Description:

This interface provides access to the compare setup for an LA module.

Namespace: Tektronix.LogicAnalyzer.TpiNet

Declaration Syntax:

Visual Basic

```
Interface ICompareSetup
```

C#

```
interface ICompareSetup
```

C++

```
__gc __interface ICompareSetup
```

Remarks:

The IChannelBit also provides the ability to specify don’t care compare masks on a channel by channel basis.

The IRecordAccess provides an interface for reading compare difference records.

2.28.1 ICompareSetup Properties

2.28.1.1 ICompareSetup.Enable

Description:

The Enable property provides the LA module with a modifiable compare enable flag.

Declaration Syntax:

Visual Basic

```
Property Enable As Boolean
```

C#

```
bool Enable { get; set;}
```

C++

```
bool get_Enable ();
void set_Enable (bool enable);
```

Arguments:

enable – desired compare enable for the module.

Exceptions Thrown:

InvalidOperationException :
 Condition: Attempt is made to set compare enable on a read only setup.
 Message: Instrument setup is read-only.

Remarks:

Changing the enable raises the CompareChanged event.

The IDataSource.IsReadOnlySetup property should be checked before attempting to modify the compare enable of an LA module.

2.28.1.2 ICompareSetup.ReferenceModule

Description:

The ReferenceModule property allows specification of a reference module for compare to the current module.

Declaration Syntax:

Visual Basic

```
Property ReferenceModule As ILAModule
```

C#

```
ILAModule ReferenceModule { get; set;}
```

C++

```
ILAModule* get_ReferenceModule ();
void set_ReferenceModule (ILAModule* module);
```

Arguments:

module – desired reference module to compare against.

Exceptions Thrown:

InvalidOperationException :
    Condition: Attempt is made to set compare reference on a read only setup.
    Message: Instrument setup is read-only.

ArgumentException :
    Condition: Attempt is made to set compare reference to an incompatible module.
    Message: Incompatible module specified for compare reference.

ArgumentNullException :
    Condition: Attempt is made to set compare reference to a null reference module.
    Message: Null module specified for compare reference.

Remarks:

Changing the reference module raises the CompareChanged event only when ICompareSetup.Enable is true.

The IDataSource.IsReadOnlySetup property should be checked before attempting to modify the compare setup.

2.28.1.3 ICompareSetup.UseReferenceRegion

Description:

The UseReferenceRegion property determines if compare is performed only for a region of the reference module samples or if all of the reference module samples are compared.

Declaration Syntax:

Visual Basic

```
Property UseReferenceRegion As Boolean
```

C#

```
bool UseReferenceRegion { get; set;}
```

C++

```
bool get_UseReferenceRegion ();
void set_UseReferenceRegion ( bool compareRegion );
```

Arguments:

compareRegion – true to compare only a region or false to compare all of reference module samples.

Exceptions Thrown:

InvalidOperationException :

Condition: Attempt is made to modify compare setup on a read only setup.
Message: Instrument setup is read-only.

Remarks:

Changing this property raises the CompareChanged event only when ICompareSetup.Enable is true.

The IDataSource.IsReadOnlySetup property should be checked before attempting to modify the compare setup.

2.28.1.4 ICompareSetup.ReferenceRegionStart

Description:

The ReferenceRegionStart property specifies where in the reference module to start comparing samples. The property includes a sample offset and anchor point (begin or trigger). An attempt to set the start to a negative Offset and RelativeToBegin will coerce the Offset to zero.

Declaration Syntax:

Visual Basic

```
Property ReferenceRegionStart As ComparePoint
```

C#

```
ComparePoint ReferenceRegionStart { get; set;}
```

C++

```
ComparePoint get_ReferenceRegionStart ();
void set_ReferenceRegionStart (ComparePoint start );
```

Arguments:

start – desired position in reference module to begin compares.

Exceptions Thrown:

InvalidOperationException :
Condition: Attempt is made to modify compare setup on a read only setup.
Message: Instrument setup is read-only.

Remarks:

Changing this property raises the CompareChanged event only when both ICompareSetup.Enable and ICompareSetup.UseReferenceRegion are true.

The IDataSource.IsReadOnlySetup property should be checked before attempting to modify the compare setup.

2.28.1.5 ICompareSetup.ReferenceRegionTotalSamples

Description:

The ReferenceRegionTotalSamples property allows access to how many samples should be compared. This property is not used when UseReferenceRegion is set to false. ReferenceRegionTotalSamples is a non-negative number and may be larger than the actual number of samples available for the modules.

Declaration Syntax:

Visual Basic

```
Property ReferenceRegionTotalSamples As Long
```

C#

```
long ReferenceRegionTotalSamples { get; set;}
```

C++

```
__int64 get_ReferenceRegionTotalSamples ();
void set_ReferenceRegionTotalSamples ( __int64 total );
```

Arguments:

total – desired number of samples to compare.

Exceptions Thrown:

InvalidOperationException :
Condition: Attempt is made to modify compare setup on a read only setup.
Message: Instrument setup is read-only.

ArgumentException :
Condition: Attempt is made to specify a negative number of samples.
Message: Invalid compare region size.

Remarks:

Changing this property raises the CompareChanged event only when both the Enable and UseReferenceRegion properties are true.

The value of ReferenceRegionTotalSamples has no affect on compare if UseReferenceRegion is false.

The IDataSource.IsReadOnlySetup property should be checked before attempting to modify the compare setup.

2.28.1.6 ICompareSetup.SourceAlignment

Description:

The SourceAlignment property specifies how the source and reference module sample numbers should be aligned for compare purposes. The SourceAlignment.RelativeTo property provides a coarse alignment by absolute sample number (RelativeToBegin) or a coarse alignment of both trigger samples (RelativeToTrigger). The SourceAlignment.Offset property provides an additional finer alignment to match earlier source samples (using a negative sample number offset) to reference samples or to match later source samples (using a positive sample number offset) to reference samples.

Declaration Syntax:

Visual Basic

```
Property SourceAlignment As ComparePoint
```

C#

```
ComparePoint SourceAlignment { get; set;}
```

C++

```
ComparePoint get_SourceAlignment ();
void set_SourceAlignment (ComparePoint align );
```

Arguments:

align – coarse alignment by begin or trigger sample and fine alignment with additional offset.

Exceptions Thrown:

InvalidOperationException :
Condition: Attempt is made to modify compare setup on a read only setup.
Message: Instrument setup is read-only.

Remarks:

Changing this property raises the CompareChanged event only when ICompareSetup.Enable is true.

The IDataSource.IsReadOnlySetup property should be checked before attempting to modify the compare setup.

2.28.2 ICompareSetup Events

2.28.2.1 ICompareSetup.CompareChanged

Description:

When a change to the compare setup is made that would alter compare results, the CompareChanged event is raised.

Declaration Syntax:

Visual Basic

```
Event CompareChanged As ObjectEventHandler
```

C#

```
event ObjectEventHandler CompareChanged;
```

C++

```
__event ObjectEventHandler CompareChanged;
```

Event Data:

The IChannelBit object is passed as a parameter.

2.29 IMemoryDepth

Description:

This interface provides the ability to set the memory depth for the next acquisition.

Namespace: Tektronix.LogicAnalyzer.TpiNet

Declaration Syntax:

Visual Basic

```
Interface IMemoryDepth
```

C#

```
interface IMemoryDepth
```

C++

```
__gc __interface IMemoryDepth
```

Remarks:

None.

2.29.1 IMemoryDepth Properties

2.29.1.1 IMemoryDepth.MemoryDepth

Description:

The MemoryDepth property allows modification of the memory depth for the next acquisition.

Declaration Syntax:

Visual Basic

```
Property MemoryDepth As Long
```

C#

```
long MemoryDepth { get; set;}
```

C++

```
__int64 get_MemoryDepth ();
void set_MemoryDepth ( __int64 depth );
```

Arguments:

depth – desired memory depth for the next acquisition.

Exceptions Thrown:

InvalidOperationException :
Condition: Attempt is made to modify a read only setup.
Message: Instrument setup is read-only.

Remarks:

The IDataSource.IsReadOnlySetup property should be checked before attempting to modify the memory depth.

When an attempt is made to set the memory depth to an invalid value, the depth will be set to the nearest valid memory depth available.

The set of valid memory depths may change based on other setup properties. This property is self-adjusting to maintain a legal value at all times.

2.29.1.2 IMemoryDepth.MaximumMemoryDepth

Description:

The read only MaximumMemoryDepth property provides the current maximum allowable memory depth for the module.

Declaration Syntax:

Visual Basic

```
ReadOnly Property MaximumMemoryDepth As Long
```

C#

```
long MaximumMemoryDepth { get;}
```

C++

```
__int64 get_MaximumMemoryDepth ();
```

Arguments:

None.

Exceptions Thrown:

None.

Remarks:

The set of valid memory depths may change based on other setup properties. This property is self-adjusting to provide the maximum legal value at all times.

2.29.1.3 IMemoryDepth.MinimumMemoryDepth

Description:

The read only MinimumMemoryDepth property provides the current minimum allowable memory depth for the module.

Declaration Syntax:

Visual Basic

```
ReadOnly Property MinimumMemoryDepth As Long
```

C#

```
long MinimumMemoryDepth { get;}
```

C++

```
__int64 get_MinimumMemoryDepth ();
```

Arguments:

None.

Exceptions Thrown:

None.

Remarks:

The set of valid memory depths may change based on other setup properties. This property is self-adjusting to provide the minimum legal value at all times.

2.30 IAcquireMode

Description:

This interface provides the ability to set the acquire mode of the LA module for the next acquisition.

Namespace: Tektronix.LogicAnalyzer.TpiNet

Declaration Syntax:

Visual Basic

```
Interface IAcquireMode
```

C#

```
interface IAcquireMode
```

C++

```
__gc __interface IAcquireMode
```

2.30.1 IAcquireMode Properties

2.30.1.1 IAcquireMode.AcquireMode

Description:

The AcquireMode property allows modification of the acquire setting for the next acquisition.

Declaration Syntax:

Visual Basic

```
Property AcquireMode As AcquireModeValue
```

C#

```
AcquireModeValue AcquireMode { get; set;}
```

C++

```
AcquireModeValue get_AcquireMode ();
void set_AcquireMode ( AcquireModeValue mode );
```

Arguments:

mode – desired acquire mode for the next acquisition.

Exceptions Thrown:

InvalidOperationException :
Condition: Attempt is made to modify a read only setup.
Message: Instrument setup is read-only.

Remarks:

The IDataSource.IsReadOnlySetup property should be checked before attempting to modify the acquire mode.

When an attempt is made to set the acquire mode to an invalid value, the acquire mode will be set to AcquireModeNormal.

The set of valid acquire modes may change based on other setup properties. This property is self-adjusting to maintain a legal value at all times.

2.31 ISamplePeriod

Description:

This interface provides the ability to set the module sample rate as a period for the next acquisition.

Namespace: Tektronix.LogicAnalyzer.TpiNet

Declaration Syntax:

Visual Basic

```
Interface ISamplePeriod
```

C#

```
interface ISamplePeriod
```

C++

```
__gc __interface ISamplePeriod
```

Remarks:

Some modules may provide distinct instances of the sample period interface corresponding to different timebases. The LA module should provide a method to return the ISamplePeriod interface associated with the MagniVu timebase.

Some modules may provide distinct instances of the sample period interface corresponding to different clock modes. While the DSO module would only be expected to provide a single instance of the ISamplePeriod, the LA module should provide a method to return distinct instances of the ISamplePeriod for the main timebase based on the clocking mode.

2.31.1 ISamplePeriod Properties

2.31.1.1 ISamplePeriod.SamplePeriod

Description:

The SamplePeriod property represents the module sample period in seconds.

Declaration Syntax:

Visual Basic

```
Property SamplePeriod As Double
```

C#

```
double SamplePeriod { get; set;}
```

C++

```
double get_SamplePeriod ();
void set_SamplePeriod ( double seconds );
```

Arguments:

`seconds` – the sample period in seconds.

Exceptions Thrown:

InvalidOperationException :
    Condition: Attempt is made to modify a read only setup.
    Message: Instrument setup is read-only.

Remarks:

The IDataSource.IsReadOnlySetup property should be checked before attempting to set this property.

When an attempt is made to set the period to an invalid value, the period will be set to the nearest valid period available.

The set of valid periods may change based on other setup properties. This property is self-adjusting to maintain a legal value at all times.

2.31.1.2 ISamplePeriod.MaximumSamplePeriod

Description:

The read only MaximumSamplePeriod property provides the current maximum allowable sample period for the module in seconds.

Declaration Syntax:

Visual Basic

```
ReadOnly Property MaximumSamplePeriod As Double
```

C#

```
double MaximumSamplePeriod { get;}
```

C++

```
double get_MaximumSamplePeriod ();
```

Arguments:

None.

Exceptions Thrown:

None.

Remarks:

The set of valid sample periods may change based on other setup properties. This property is self-adjusting to provide the maximum legal value at all times.

2.31.1.3 ISamplePeriod.MinimumSamplePeriod

Description:

The read only MinimumSamplePeriod property provides the current minimum allowable sample period for the module in seconds.

Declaration Syntax:

Visual Basic

```
ReadOnly Property MinimumSamplePeriod As Double
```

C#

```
double MinimumSamplePeriod { get;}
```

C++

```
double get_MinimumSamplePeriod ();
```

Arguments:

None.

Exceptions Thrown:

None.

Remarks:

The set of valid sample periods may change based on other setup properties. This property is self-adjusting to provide the minimum legal value at all times.

2.32 IClockMode

Description:

This interface provides the ability to set the clock mode of the LA module for the next acquisition.

Namespace: Tektronix.LogicAnalyzer.TpiNet

Declaration Syntax:

Visual Basic

```
Interface IClockMode
```

C#

```
interface IClockMode
```

C++

```
__gc __interface IClockMode
```

Remarks:

None.

2.32.1 IClockMode Properties

2.32.1.1 IClockMode.ClockMode

Description:

The ClockMode property represents the module clock mode for the next acquisition.

Declaration Syntax:

Visual Basic

```
Property ClockMode As ClockModeValue
```

C#

```
ClockModeValue ClockMode { get; set;}
```

C++

```
ClockModeValue get_ClockMode ();
void set_ClockMode ( ClockModeValue mode );
```

Arguments:

`mode` – the clock mode for the next acquisition.

Exceptions Thrown:

InvalidOperationException :
Condition: Attempt is made to modify a read only setup.
Message: Instrument setup is read-only.

Remarks:

The IDataSource.IsReadOnlySetup property should be checked before attempting to set this property.

2.33 ICorrelator

Description:

This interface provides the ability to time correlate sample numbers from multiple data sources and data sets.

Namespace: Tektronix.LogicAnalyzer.TpiNet

Declaration Syntax:

Visual Basic

```
Interface ICorrelator Inherits IValidity
```

C#

```
interface ICorrelator : IValidity
```

C++

```
__gc __interface ICorrelator : public IValidity
```

Remarks:

The correlator is used by adding correlatable items to it with AddItem(). Once the items have been added, a number of methods are available to obtain information about how their samples are arranged in time.

All time values used as parameters to ICorrelator methods or as return data within a CorrelatorPoint structure include time offset adjustments for Expansion/Mainframe, time distance from System Trigger time, user adjustable time alignment of data sources, and user adjustable time alignment of individual DSO and External Oscilloscope channels. The overall adjustment from the original timestamp value of the data source is available via IAcquisitionData.CorrelationTimeOffset. Additional offset applied to a DSO or External Oscilloscope channel is available via IChannelGroup.TimeOffset.

The correlation model associates a correlated sequence number with the sample of exactly one of the items in the correlation set. In other words, correlated items do not share a correlated sequence number. When multiple correlated items share a timestamp value, the correlator uses a ranking scheme to break the tie. The ranking is determined by the order in which items were added to the correlator (see the AddItem() method).

The correlator only uses LA samples that survive the sample suppression. The module sample numbers for a module with suppression are consecutively numbered with no gaps in sample numbers. Sample numbers for an LA module with suppression in effect are not guaranteed to match the original unsuppressed sample numbers or to remain the same when the suppression definition is changed.

Data filtering for LA modules is an independent operation that should be applied by the client on the output of the correlation results.

Clients should call Dispose() on the ICorrelator instance when finished to allow the TLA application to free resources associated with the instance. When the TLA application exits, it also calls Dispose() on ICorrelator instances that have not already been disposed.

2.33.1 ICorrelator Methods

2.33.1.1 ICorrelator.AddItem

Description:

This method is used to add a new correlation item to the correlator. A correlation item specifies the IStandardDataSource and DataSetValue to be included in correlation results. It may optionally include a IChannelGroup. The IChannelGroup is used to specifiy a DSO or External Oscilloscope group that has been independently time-aligned relative to the rest of the instrument data.

Declaration Syntax:

Visual Basic

```
Function AddItem (source As IStandardDataSource,
                  dset As DataSetValue,
                  group As IChannelGroup) As Integer
```

C#

```
int AddItem (IStandardDataSource source,
             DataSetValue dset,
             IChannelGroup group)
```

C++

```
int AddItem (IStandardDataSource *source,
             DataSetValue dset,
             IChannelGroup *group)
```

Arguments:

source – the instrument data source to be correlated.
dset – the instrument data set to be correlated.
group – an optional DSO or External Oscilloscope group with custom time-alignment.

Return Value:

A non-negative key for the item is returned on success. This key should be used as a handle to the item in subsequent correlation calls. On failure, a negative value is returned.

Exceptions Thrown:

None.

Remarks:

The order in which items are added to the correlator determines how samples are ordered in the correlation sequence in the case when two or more items have identical timestamps. Items added earlier have their samples ordered earlier than other items with identical timestamps.

Only Main and MagniVu data set values are allowed. Any data set value other than MagniVu will be coerced to Main.

If the specified item has already been added to the correlator, the same item handle will be returned for both instances.

The correlator does not manage removing items from the collection that have been deleted from their respective channel setups. When data sources or groups are disposed, a new ICorrelation object should be created and the surviving data sources and groups should be added to it.

2.33.1.2 ICorrelator.NumberOfPoints

Description:

This method determines the total number of correlated points (sequences) among the items added to the set. This is the sum of the number of samples for all items in the set.

Declaration Syntax:

Visual Basic

```
Function NumberOfPoints () As Long
```

C#

```
long NumberOfPoints ()
```

C++

```
__int64 NumberOfPoints ()
```

Arguments:

None.

Return Value:

The total number of distinct sample numbers known to the correlator.

Exceptions Thrown:

None.

Remarks:

None.

2.33.1.3 ICorrelator.ItemSamplePosition

Description:

This method converts an item sample into a time-correlated sequence number. The sequence numbering is zero based.

Declaration Syntax:

Visual Basic

```
Function ItemSamplePosition (item As Integer, sample As Long) As Long
```

C#

```
long ItemSamplePosition (int item, long sample)
```

C++

```
__int64 ItemSamplePosition (int item, long sample)
```

Arguments:

item – handle to an item previously added by AddItem().
sample – the sample number of the item.

Return Value:

On success, this returns a non-negative number representing the order of the item sample within the correlation sequence. On failure, this returns a negative number.

Exceptions Thrown:

None.

Remarks:

This method returns a negative number if the item is not recognized or if the item has no such sample number.

The earliest sample from all data sources is sequence number zero. If this method determined that the sample of interest was third in the time ordering sequence of samples, it would return 2.

2.33.1.4 ICorrelator.ItemBeginPoint

Description:

This method returns correlation information about the first sample for the designated item. The Time property of the return value is typically the data of interest by callers.

Declaration Syntax:

Visual Basic

```
Function ItemBeginPoint (item As Integer) As CorrelatorPoint
```

C#

```
CorrelatorPoint ItemBeginPoint (int item)
```

C++

```
CorrelatorPoint ItemBeginPoint (int item)
```

Arguments:

item – handle to an item previously added by AddItem().

Return Value:

A CorrelatorPoint data structure describing the first acquisition sample of the item.

Exceptions Thrown:

None.

Remarks:

The caller should check the CorrelatorPoint.IsValid member of the return value. This will be false on error when the item is not recognized or the associated data source has no samples.

2.33.1.5 ICorrelator.ItemEndPoint

Description:

This method returns correlation information about the ending sample for the designated item. The Time property of the return value is typically the data of interest by callers.

Declaration Syntax:

Visual Basic

```
Function ItemEndPoint (item As Integer) As CorrelatorPoint
```

C#

```
CorrelatorPoint ItemEndPoint (int item)
```

C++

```
CorrelatorPoint ItemEndPoint (int item)
```

Arguments:

item – handle to an item previously added by AddItem().

Return Value:

A CorrelatorPoint data structure describing the ending acquisition sample of the item.

Exceptions Thrown:

None.

Remarks:

The caller should check the CorrelatorPoint.IsValid member of the return value. This will be false on error when the item is not recognized or the associated data source has no samples.

2.33.1.6 ICorrelator.BeginPoint

Description:

This method returns correlation information about the earliest sample among all items. The Item and Time properties of the return value are typically the data of interest by callers.

Declaration Syntax:

Visual Basic

```
Function BeginPoint () As CorrelatorPoint
```

C#

```
CorrelatorPoint BeginPoint ()
```

C++

```
CorrelatorPoint BeginPoint ()
```

Arguments:

None.

Return Value:

A CorrelatorPoint data structure describing the earliest acquisition sample among all items.

Exceptions Thrown:

None.

Remarks:

The caller should check the CorrelatorPoint.IsValid member of the return value. This will be false on error when no items in the set contain acquisition samples.

2.33.1.7 ICorrelator.EndPoint

Description:

This method returns correlation information about the latest sample among all items. The Item and Time properties of the return value are typically the data of interest by callers.

Declaration Syntax:

Visual Basic

```
Function EndPoint () As CorrelatorPoint
```

C#

```
CorrelatorPoint EndPoint ()
```

C++

```
CorrelatorPoint EndPoint ()
```

Arguments:

None.

Return Value:

A CorrelatorPoint data structure describing the latest acquisition sample among all items.

Exceptions Thrown:

None.

Remarks:

The caller should check the CorrelatorPoint.IsValid member of the return value. This will be false on error when no items in the set contain acquisition samples.

2.33.1.8 ICorrelator.ItemSample

Description:

This method returns correlation information about the designated sample for the designated item. The Time property of the return value is typically the data of interest by callers.

Declaration Syntax:

Visual Basic

```
Function ItemSample (item As Integer, sample As Long) As
CorrelatorPoint
```

C#

```
CorrelatorPoint ItemSample (int item, long sample)
```

C++

```
CorrelatorPoint ItemSample (int item, __int64 sample)
```

Arguments:

item – handle to an item previously added by AddItem().
sample – sample number of the item.

Return Value:

A CorrelatorPoint data structure describing the sample of the item.

Exceptions Thrown:

None.

Remarks:

The caller should check the CorrelatorPoint.IsValid member of the return value. This will be false on error when the item is not recognized or the associated data source has no such sample.

2.33.1.9 ICorrelator.ItemAtOrAfterTime

Description:

This method returns correlation information about the earliest sample belonging to the designated item that occurs at or after the designated time. The Sample and Time properties of the return value are typically the data of interest by callers.

Declaration Syntax:

Visual Basic

```
Function ItemAtOrAfterTime (item As Integer, time As Decimal) As
CorrelatorPoint
```

C#

```
CorrelatorPoint ItemAtOrAfterTime (int item, decimal time)
```

C++

```
CorrelatorPoint ItemAtOrAfterTime (int item, Decimal time)
```

Arguments:

item – handle to an item previously added by AddItem().
time – the earliest time of interest.

Return Value:

A CorrelatorPoint data structure describing the sample of the item.

Exceptions Thrown:

None.

Remarks:

The caller should check the CorrelatorPoint.IsValid member of the return value. This will be false on error when the item is not recognized or the associated data source has no such sample.

2.33.1.10 ICorrelator.ItemAtOrBeforeTime

Description:

This method returns correlation information about the latest sample belonging to the designated item that occurs at or before the designated time. The Sample and Time properties of the return value are typically the data of interest by callers.

Declaration Syntax:

Visual Basic

```
Function ItemAtOrBeforeTime (item As Integer, time As Decimal) As
CorrelatorPoint
```

C#

```
CorrelatorPoint ItemAtOrBeforeTime (int item, decimal time)
```

C++

```
CorrelatorPoint ItemAtOrBeforeTime (int item, Decimal time)
```

Arguments:

item – handle to an item previously added by AddItem().
time – the latest time of interest.

Return Value:

A CorrelatorPoint data structure describing the sample of the item.

Exceptions Thrown:

None.

Remarks:

The caller should check the CorrelatorPoint.IsValid member of the return value. This will be false on error when the item is not recognized or the associated data source has no such sample.

2.33.1.11 ICorrelator.AtOrAfterTime

Description:

This method returns correlation information about the earliest sample belonging to any of the items that occurs at or after the designated time. The Item, Sample, and Time properties of the return value are typically the data of interest by callers.

Declaration Syntax:

Visual Basic

```
Function ItemAtOrAfterTime (time As Decimal) As CorrelatorPoint
```

C#

```
CorrelatorPoint ItemAtOrAfterTime (decimal time)
```

C++

```
CorrelatorPoint ItemAtOrAfterTime (Decimal time)
```

Arguments:

time – the earliest time of interest.

Return Value:

A CorrelatorPoint data structure describing the sample of the item.

Exceptions Thrown:

None.

Remarks:

The caller should check the CorrelatorPoint.IsValid member of the return value. This will be false on error when no samples exist at or after the specified time.

2.33.1.12 ICorrelator.AtOrBeforeTime

Description:

This method returns correlation information about the latest sample belonging to any of the items that occurs at or before the designated time. The Item, Sample, and Time properties of the return value are typically the data of interest by callers.

Declaration Syntax:

Visual Basic

```
Function ItemAtOrBeforeTime (time As Decimal) As CorrelatorPoint
```

C#

```
CorrelatorPoint ItemAtOrBeforeTime (decimal time)
```

C++

```
CorrelatorPoint ItemAtOrBeforeTime (Decimal time)
```

Arguments:

time – the earliest time of interest.

Return Value:

A CorrelatorPoint data structure describing the sample of the item.

Exceptions Thrown:

None.

Remarks:

The caller should check the CorrelatorPoint.IsValid member of the return value. This will be false on error when no samples exist at or before the specified time.

2.33.1.13 ICorrelator.NextSample

Description:

This method finds the correlation point belonging to the designated item and sample and then returns correlation information about the sample that follows. The next sample may belong to a different item. The Item, Sample, and Time properties of the return value are typically the data of interest by callers.

Declaration Syntax:

Visual Basic

```
Function NextSample (item As Integer, sample As Long) As
CorrelatorPoint
```

C#

```
CorrelatorPoint NextSample (int item, long sample)
```

C++

```
CorrelatorPoint NextSample (int item, __int64 sample)
```

Arguments:

item – handle to an item previously added by AddItem().
sample – sample number prior to the area of interest.

Return Value:

A CorrelatorPoint data structure describing the sample of the item.

Exceptions Thrown:

None.

Remarks:

The caller should check the CorrelatorPoint.IsValid member of the return value. This will be false on error when the item is not recognized, the associated data source has no such sample, or no samples exist from any items after that point.

2.33.1.14 ICorrelator.PriorSample

Description:

This method finds the correlation point belonging to the designated item and sample and then returns correlation information about the preceding sample. The preceding sample may belong to a different item. The Item, Sample, and Time properties of the return value are typically the data of interest by callers.

Declaration Syntax:

Visual Basic

```
Function PriorSample (item As Integer, sample As Long) As
CorrelatorPoint
```

C#

```
CorrelatorPoint PriorSample (int item, long sample)
```

C++

```
CorrelatorPoint PriorSample (int item, __int64 sample)
```

Arguments:

item – handle to an item previously added by AddItem().
sample – sample number subsequent to the area of interest.

Return Value:

A CorrelatorPoint data structure describing the sample of the item.

Exceptions Thrown:

None.

Remarks:

The caller should check the CorrelatorPoint.IsValid member of the return value. This will be false on error when the item is not recognized, the associated data source has no such sample, or no samples exist from any items prior to that point.

2.33.2 ICorrelator Events

2.33.2.1 ICorrelator.SequencingChanged

Description:

When the sequencing of data changes, the SequencingChanged event is raised. This could be due to new or invalidated acquisition data, changes in the LA sample suppression, or to changes in the time alignment settings of the data source or DSO channel.

Declaration Syntax:

Visual Basic

```
Event SequencingChanged As ObjectEventHandler
```

C#

```
event ObjectEventHandler SequencingChanged;
```

C++

```
__event ObjectEventHandler SequencingChanged;
```

Event Data:

The ICorrelator object is passed as a parameter.

2.34 ILATrigger

Description:

This is the IDataSourceTrigger-based interface for all logic analyzer module trigger definitions.

Namespace: Tektronix.LogicAnalyzer.Common

Declaration Syntax:

Visual Basic

```
Interface ILATrigger
        Inherits IDataSourceTrigger
```

C#

```
interface ILATrigger : IDataSourceTrigger
```

C++

```
__gc __interface ILATrigger : public IDataSourceTrigger
```

Remarks:

The purpose of this interface is to allow methods that deal with trigger properties common to all TLA logic analyzer modules.

2.34.1 ILATrigger Properties

2.34.1.1 ILATrigger.IsMagniVuReadOnly

Description:

Indicates whether the MagniVu trigger properties are read-only for the current module.

Declaration Syntax:

Visual Basic

```
ReadOnly Property IsMagniVuReadOnly As Boolean
```

C#

```
bool IsMagniVuReadOnly { get; }
```

C++

```
bool get_IsMagniVuReadOnly ()
```

Arguments:

None.

Exceptions Thrown:

None.

Remarks:

A value of true indicates that the MagniVu trigger position is a read-only property.

Examples:

C# code:

```
//
// Examples of using ILATrigger properties and methods.
//
public bool TestLATrigger( ILATrigger laTrigger )
{
        // Start off from defaulted trigger.
    laTrigger.Default();

    bool isMagniVuReadOnly = laTrigger.IsMagniVuReadOnly;
        long minMagniVuPosition = laTrigger.MinimumMagniVuPosition;
        long maxMagniVuPosition = laTrigger.MaximumMagniVuPosition;
    double[] periods = laTrigger.AvailableMagniVuSamplePeriods;
    bool supported = laTrigger.IsForceMainPrefillSupported;
    laTrigger.ForceMainPrefill = true;

        ILATriggerStorage triggerStorage = laTrigger.GetGlobalStorage();
        GlobalStorageType storageType = triggerStorage.GlobalStorage;

        ILATriggerState triggerState = laTrigger.GetLAState(0);
        triggerState.Label = "Testing";

    long position = laTrigger.Position;
    long minPosition = laTrigger.MinimumPosition;
    long maxPosition = laTrigger.MaximumPosition;
    int numStates = laTrigger.NumberOfStates;
    bool canAdd1 = laTrigger.CanAddState();
    bool canAdd2 = laTrigger.CanAddState(triggerState);
    laTrigger.InsertState( 0, triggerState);
    laTrigger.RemoveState(0);
    laTrigger.GetState( 0 );

        ////
        // This structure is defined elsewhere in the code.
    // public MagnivuStorage( long period, long position )
    // {
    //              samplePeriod = period;
    //              maxPosition = position;
    // }
    //

        // Array of MagniVu sample period / max trigger position pairs.
        // These values are taken from the TLA on-line help, but they
        // can also be queried for from AvailableMagniVuSamplePeriods.
        MagnivuStorage[] mvStoragePairs =
        {
        new MagnivuStorage(125, 58),
        new MagnivuStorage(250,79),
        new MagnivuStorage(500,89),
        new MagnivuStorage(1000,94)
    };

        // Test all pairs.
        foreach ( MagnivuStorage storage in mvStoragePairs )
        {
                laTrigger.MagniVuSamplePeriod = storage.samplePeriod;
                laTrigger.MagniVuPosition = 0;
                laTrigger.MagniVuPosition = storage.maxPosition;
        }

        ////
        // Get list of channels to set/get.
        IChannelBit[] channels = TestUtils.GetModuleChannels(laModule);

        // Create lists of trigger violation setupValues and holdValues
        // to match the channel list.
        long[] setupValues = new long[channels.Length];
        long[] holdValues  = new long[channels.Length];

        for ( int idx = 0; idx < channels.Length; idx++ )
        {
                setupValues[idx] = (idx % 2 == 0) ? 125 : 0;
                holdValues[idx]  = (idx % 2 == 0) ? 0 : 125;
```

```
		}

		laTrigger.SetSetupHoldViolationEnable( channels, true );
		laTrigger.SetSetupHoldViolation( channels, setupValues, holdValues );

		long[] retSetupValues = new long[channels.Length];
		long[] retHoldValues  = new long[channels.Length];

		laTrigger.GetSetupHoldViolation(channels, retSetupValues, retHoldValues);

		////
		// Register for ILATrigger events.
		RegisterForLATriggerEvents();

		// Generating a MagniVuPositionChanged event.
	laTrigger.MagniVuPosition = laTrigger.MagniVuPosition + 1;

		// Generating a ForceMainPrefill event.
		laTrigger.ForceMainPrefill = ! laTrigger.ForceMainPrefill;

		// Generating a PositionChanged event.
	laTrigger.Position = 39;

		// Generating a NumberOfStatesChanged event.
	laTrigger.AddState(state);
}


//////////////////////////////////////////////////////////////////////////
// RegisterLATriggerEvents - register for LATrigger events
//
// Note that remoters have to be used in order to work with
// remote clients.  I.e, the remote clients cannot do normal event
// delegate registration in TPI.NET.  Although remoters are not needed
// for plugins, plugins do not have to adopt a different event model.
//
// These are defined elsewhere in the code:
//		private ObjectEventRemoter m_magniVuPositionChangedObjectEventRemoter;
//		private ObjectEventRemoter m_forceMainPrefillChangedObjectEventRemoter;
//		private ObjectEventRemoter m_positionChangedObjectEventRemoter;
//		private ObjectEventRemoter m_numberOfStatesChangedObjectEventRemoter;
//
private void RegisterLATriggerEvents( ILATrigger trigger )
{
		//
		// Register for the MagniVuPositionChanged event.
		//
		m_magniVuPositionChangedObjectEventRemoter = new ObjectEventRemoter();
		m_magniVuPositionChangedObjectEventRemoter.RemoteEventOccurred +=
		new ObjectEventHandler( HandleMagniVuPositionChanged );
		trigger.MagniVuPositionChanged += new ObjectEventHandler(
		m_magniVuPositionChangedObjectEventRemoter.OnRemoteEvent );

		//
		// Register for the ForceMainPrefillChanged event.
		//
```

```
        m_forceMainPrefillChangedObjectEventRemoter = new ObjectEventRemoter();
        m_forceMainPrefillChangedObjectEventRemoter.RemoteEventOccurred +=
        new ObjectEventHandler( HandleForceMainPrefillChanged );
    trigger.ForceMainPrefillChanged += new ObjectEventHandler(
        m_forceMainPrefillChangedObjectEventRemoter.OnRemoteEvent );

        //
        // Register for the PositionChanged event.
        //
        m_positionChangedObjectEventRemoter = new ObjectEventRemoter();
    m_positionChangedObjectEventRemoter.RemoteEventOccurred +=
        new ObjectEventHandler( HandlePositionChanged );
    trigger.PositionChanged += new ObjectEventHandler(
        m_positionChangedObjectEventRemoter.OnRemoteEvent );

        //
        // Register for the NumberOfStatesChanged event.
        //
        m_numberOfStatesChangedObjectEventRemoter = new ObjectEventRemoter();
        m_numberOfStatesChangedObjectEventRemoter.RemoteEventOccurred +=
        new ObjectEventHandler( HandleNumberOfStatesChanged );
        trigger.NumberOfStatesChanged += new ObjectEventHandler(
        m_numberOfStatesChangedObjectEventRemoter.OnRemoteEvent );
}

/////////////////////////////////////////////////////////////////////////
// UnregisterLATriggerEvents - unregister remoter events
//
private void UnregisterLATriggerEvents( ILATrigger trigger )
{
        trigger.MagniVuPositionChanged -=
        new ObjectEventHandler (
    m_magniVuPositionChangedObjectEventRemoter.OnRemoteEvent );
        m_magniVuPositionChangedObjectEventRemoter.RemoteEventOccurred -=
        new ObjectEventHandler( HandleMagniVuPositionChanged );
        m_magniVuPositionChangedObjectEventRemoter = null;

        trigger.ForceMainPrefillChanged -=
        new ObjectEventHandler (
    m_forceMainPrefillChangedObjectEventRemoter.OnRemoteEvent );
        m_forceMainPrefillChangedObjectEventRemoter.RemoteEventOccurred -=
        new ObjectEventHandler( HandleForceMainPrefillChanged );
        m_forceMainPrefillChangedObjectEventRemoter = null;

        trigger.PositionChanged -=
        new ObjectEventHandler ( m_positionChangedObjectEventRemoter.OnRemoteEvent );
        m_positionChangedObjectEventRemoter.RemoteEventOccurred -=
        new ObjectEventHandler( HandlePositionChanged );
        m_positionChangedObjectEventRemoter = null;

        trigger.NumberOfStatesChanged -=
        new ObjectEventHandler (
    m_numberOfStatesChangedObjectEventRemoter.OnRemoteEvent );
        m_numberOfStatesChangedObjectEventRemoter.RemoteEventOccurred -=
        new ObjectEventHandler( HandleNumberOfStatesChanged );
        m_numberOfStatesChangedObjectEventRemoter = null;
```

```
}

////////////////////////////////////////////////////////////////////////
// Handlers for the various events.
//
private
void HandleMagniVuPositionChanged( object sender, ObjectEventArgs args )
{
	// handle HandleMagniVuPositionChanged event
}

private
void HandleForceMainPrefillChanged( object sender, ObjectEventArgs args )
{
	// handle HandleForceMainPrefillChanged event
}

private
void HandlePositionChanged( object sender, ObjectEventArgs args )
{
	// handle HandlePositionChanged event
}

private
void NumberOfStatesChanged ( object sender, ObjectEventArgs args )
{
	// handle NumberOfStatesChanged event
}
```

2.34.1.2 ILATrigger.MinimumMagniVuPosition

Description:

The minimum percentage offset, from the beginning of acquired MagniVu data, that the data sample associated with the MagniVu trigger event can be stored.

Declaration Syntax:

Visual Basic

```
ReadOnly Property MinimumMagniVuPosition As Long
```

C#

```
long MinimumMagniVuPosition { get; }
```

C++

```
__int64 get_MinimumMagniVuPosition ();
```

Arguments:

None.

Exceptions Thrown:

None.

Remarks:

The minimum available MagniVu trigger position, expressed as a percentage of acquired MagniVu memory samples, may be greater than 0%.

Examples:

```
See example for ILATrigger.IsMagniVuReadOnly.
```

2.34.1.3 ILATrigger.MaximumMagniVuPosition

Description:

The maximum percentage offset, from the beginning of acquired MagniVu data, that the data sample associated with the MagniVu trigger event can be stored.

Declaration Syntax:

Visual Basic

```
ReadOnly Property MaximumMagniVuPosition As Long
```

C#

```
long MaximumMagniVuPosition { get; }
```

C++

```
__int64 get_MaximumMagniVuPosition ();
```

Arguments:

None.

Exceptions Thrown:

None.

Remarks:

The maximum available MagniVu trigger position, expressed as a percentage of acquired MagniVu memory samples, may be less than 100%.

Examples:

```
See example for ILATrigger.IsMagniVuReadOnly.
```

2.34.1.4 ILATrigger.MagniVuPosition

Description:

The percentage offset, from the beginning of acquired MagniVu data, to store the data sample associated with the LA's MagniVu trigger event.

Declaration Syntax:

Visual Basic

```
Property MagniVuPosition As Long
```

C#

```
long MagniVuPosition { set; get; }
```

C++

```
__int64 get_MagniVuPosition ();
void set_MagniVuPosition (__int64 position );
```

Arguments:

position – a value between 0 and 100 that specifies the percentage offset from the beginning of acquired data at which the trigger-event sample should be stored.

Exceptions Thrown:

InvalidOperationException :
Condition: Attempting to set a read-only property.
Message: MagniVu position is read-only for this LA module.

ArgumentOutOfRangeException:
Condition: MagniVu trigger position percentage is out of bounds.
Message: The requested MagniVu position percentage is outside the range currently possible.

Remarks:

Not all LA modules have a programmable MagniVu trigger position. Attempting to set the position on such modules will throw a non-fatal exception. Check the ILATrigger.IsMagniVuReadOnly property to determine whether the MagniVu trigger position can be set.

The range of MagniVu trigger positions that can be set can vary, depending on other settings in the LA module's setup. Query the ILATrigger.MinimumMagniVuPosition and ILATrigger.MaximumMagniVuPosition properties to determine the currently valid range.

Examples:

```
See example for ILATrigger.IsMagniVuReadOnly.
```

2.34.1.5 ILATrigger.AvailableMagniVuSamplePeriods

Description:

An array of doubles that represent all currently available MagniVu sample periods, expressed in picoseconds.

Declaration Syntax:

Visual Basic

```
ReadOnly Property AvailableMagniVuSamplePeriods As ArrayList
```

C#

```
ArrayList AvailableMagniVuSamplePeriods { get; }
```

C++

```
ArrayList* get_AvailableMagniVuSamplePeriods()
```

Arguments:

None.

Exceptions Thrown:

None.

Remarks:

Attempting to change the content of the property will throw an exception.

Examples:

```
See example for ILATrigger.IsMagniVuReadOnly.
```

2.34.1.6 ILATrigger.MagniVuSamplePeriod

Description:

The period, in picoseconds, between stored MagniVu samples.

Declaration Syntax:

Visual Basic

```
Property MagniVuSamplePeriod As Double
```

C#

```
Double MagniVuSamplePeriod { set; get; }
```

C++

```
Double get_MagniVuSamplePeriod ();
void set_MagniVuSamplePeriod ( Double period );
```

Arguments:

`period` – the interval, in picoseconds, between stored MagniVu samples.

Exceptions Thrown:

InvalidOperationException :
    Condition: Attempting to set a read-only property.
    Message: MagniVu sample period is read-only for the LA module.

ArgumentException :
    Condition: Specified period isn't supported by the LA module.
    Message: The requested MagniVu sample period is unsupported by the LA module.

Remarks:

Not all LA modules have a programmable MagniVu sample period. Attempting to set the position on such modules will throw a non-fatal exception. Check the ILATrigger.IsMagniVuReadOnly property to determine whether the MagniVu sample period can be set.

The number of MagniVu sample periods available varies, depending on other settings in the LA module's setup. Query the array of sample periods in the ILATrigger.AvailableMagniVuSamplePeriods array property to determine what periods can be used.

Examples:

```
See example for ILATrigger.IsMagniVuReadOnly.
```

2.34.1.7 ILATrigger.IsForceMainPrefillSupported

Description:

Indicates whether the forced prefill of pre-trigger samples in main acquisition memory supported by the LA module.

Declaration Syntax:

Visual Basic

```
ReadOnly Property IsForceMainPrefillSupported As Boolean
```

C#

```
bool IsForceMainPrefillSupported { get; }
```

C++

```
bool get_IsForceMainPrefillSupported ()
```

Arguments:

None.

Exceptions Thrown:

None.

Remarks:

A value of true indicates that the forced prefill of pre-trigger samples is supported.

Examples:

```
See example for ILATrigger.IsMagniVuReadOnly.
```

2.34.1.8 ILATrigger.ForceMainPrefill

Description:

Controls whether pre-trigger samples must be stored in main memory before a trigger can occur.

Declaration Syntax:

Visual Basic

```
Property ForceMainPrefill As Boolean
```

C#

```
bool ForceMainPrefill { set; get; }
```

C++

```
bool get_ForceMainPrefill ();
void set_ForceMainPrefill ( bool enable );
```

Arguments:

`enable` – a value of true enables forced prefill; false disables it.

Exceptions Thrown:

InvalidOperationException :
Condition: Attempting to set an unsupported property.
Message: Force Main Prefill is unsupported for this LA module.

Remarks:

Not all LA modules have a force main prefill feature. Attempting to set or read the property in such a case throws a non-fatal exception. Check the ILATrigger.IsForceMainPrefillSupported property to determine whether ILATrigger.ForceMainPrefill can be accessed.

Examples:

```
See example for ILATrigger.IsMagniVuReadOnly.
```

2.34.2 ILATrigger Methods

2.34.2.1 ILATrigger.GetGlobalStorage

Description:

Returns the ILATriggerStorage interface that manages the LA trigger definition’s global storage.

Declaration Syntax:

Visual Basic

```
Function GetGlobalStorage () As ILATriggerStorage
```

C#

```
ILATriggerStorage GetGlobalStorage ()
```

C++

```
ILATriggerStorage * GetGlobalStorage ()
```

Arguments:

None.

Return Value:

The interface to the LA trigger definition’s global storage settings.

Exceptions Thrown:

None.

Remarks:

None.

Examples:

```
See example for ILATriggerStorage.
```

2.34.2.2 ILATrigger.GetLAState

Description:

Returns the specified LA trigger state from the trigger definition.

Declaration Syntax:

Visual Basic

```
Function GetLAState (index As Int32 ) As ILATriggerState
```

C#

```
ILATriggerState GetLAState ( Int32 index )
```

C++

```
ILATriggerState * GetLAState ( Int32 index )
```

Arguments:

`index` – zero-based index of the LA trigger state to return.

Return Value:

The interface to the specified LA trigger state.

Exceptions Thrown:

ArgumentOutOfRangeException:
    Condition: Out-of-bounds state index.
    Message: The state index does not represent a valid trigger state location.

Remarks:

This is a thin wrapper for casting the return value of IDataSourceTrigger.GetState() to the appropriate LA trigger state interface.

Examples:

```
See example for ILATriggerState.
```

2.34.2.3 ILATrigger.SetSetupHoldViolation

Description:

Assigns channel setup and hold values for timing violation events and storage.

Declaration Syntax:

Visual Basic

```
Sub SetSetupHoldViolation ( channels As TlaCommon.IChannelBit(),
                              setupValues As Int64(),
                              holdValues As Int64 () )
```

C#

```
void  SetSetupHoldViolation( TlaCommon.IChannelBit[] channels,
                              Int64 [] setupValues,
                              Int64 [] holdValues )
```

C++

```
void SetSetupHoldViolation ( IChannelBit* channels[],
                              Int64 setupValues[],
                              Int64 holdValues[] )
```

Arguments:

`channels` – an array of IChannelBits.

`setupValues` – an array of setup times, expressed in picoseconds.

`holdValues` – an array hold times, expressed in picoseconds.

Return Value:

None.

Exceptions Thrown:

ArgumentException:
 Condition: Channel array is empty.
 Message: No channels specified. Array is empty.

ArgumentException:
 Condition: Both value arrays are empty.
 Message: No setup or hold values specified. Both arrays are empty.

ArgumentException:
 Condition: The number of setup values is different than the number of channels specified.
 Message: Number of setup values does not match number of channels.

ArgumentException:
Condition: The number of hold values is different than the number of channels specified.
Message: Number of hold values does not match number of channels.

ArgumentException:
Condition: Illegal IChannelBit in the channels array.
Message: Specified channel is not in the current instrument.

ArgumentException:
Condition: The combination of setup and hold values specified for one of the channels in the array is not supported by the hardware.
Message: Setup and hold combination for specified channel is invalid.

ArgumentException:
Condition: The setup value specified for one of the channels in the array is not supported by the hardware.
Message: Setup value for specified channel is invalid.

ArgumentException:
Condition: The hold value specified for one of the channels in the array is not supported by the hardware.
Message: Hold value for specified channel is invalid.

Remarks:

For each item in the channels[] parameter, the method uses the corresponding picosecond value in the setupValues[] and holdValues[] parameters to define the setup and hold window to use when detecting timing violation for the given channel.

Either the setupValues[] array or the holdValues[] array can be empty, but not both.

If an array is not empty, it must contain the same number of items as the channels[] array.

The values used are 64-bit integers that specify the number of picoseconds. The values can be negative.

Examples:

```
See example for ILATrigger.IsMagniVuReadOnly.
```

2.34.2.4 ILATrigger.GetSetupHoldViolation

Description:

Returns arrays of channel setup and hold values currently used for timing violation events and storage.

Declaration Syntax:

Visual Basic

```
Sub GetSetupHoldViolation ( [in] channels As TlaCommon.IChannelBit(),
                              [out] setupValues As Int64(),
                              [out] holdValues As Int64 () )
```

C#

```
void  GetSetupHoldViolation([in] TlaCommon.IChannelBit[] channels,
                              [out] Int64 [] setupValues,
                              [out] Int64 [] holdValues )
```

C++

```
void GetSetupHoldViolation ([in] IChannelBit* channels[],
                              [out] Int64 setupValues[],
                              [out] Int64 holdValues[] )
```

Arguments:

`channels` – an array of IChannelBits.

`setupValues` – an array of setup times, expressed in picoseconds.

`holdValues` – an array hold times, expressed in picoseconds.

Return Value:

The contents of setupValues[] and holdValues[] parameters are updated .

Exceptions Thrown:

ArgumentException:
  Condition: Channel array is empty.
  Message: No channels specified. Array is empty.

ArgumentException:
  Condition: Illegal IChannelBit in the channels array.
  Message: Specified channel is not in the current instrument.

Remarks:

For each item in the channels[] parameter, the method retrieves the picosecond values currently assigned to the channel for setup and hold timing violation detection, and places them at the corresponding locations in the setupValues[] and holdValues[] parameters.

The method allocates space in both the value arrays to correspond to the number of entries in the channels[] parameter.

The values returned are 64-bit integers that specify the number of picoseconds. The values can be negative.

Examples:

```
See example for ILATrigger.IsMagniVuReadOnly.
```

2.34.2.5 ILATrigger.SetSetupHoldViolationEnable

Description:

Controls which channels are enabled to monitor setup and hold timing violations.

Declaration Syntax:

Visual Basic

```
Sub SetSetupHoldViolationEnable ( channels As TlaCommon.IChannelBit(),
                              enable As Boolean )
```

C#

```
void  SetSetupHoldViolationEnable ( TlaCommon.IChannelBit[] channels,
                              bool enable )
```

C++

```
void SetSetupHoldViolationEnable ( IChannelBit* channels[],
                              bool enable )
```

Arguments:

`channels` – an array of IChannelBits.

`enable` – the enable state to set all channels specified in the channels[] parameter.

Return Value:

None.

Exceptions Thrown:

ArgumentException:
  Condition: Channel array is empty.
  Message: No channels specified. Array is empty.

ArgumentException:
  Condition: Illegal IChannelBit in the channels array.
  Message: Specified channel is not in the current instrument.

Remarks:

The method sets the enable state of all channels contained in the channels[] parameter to the value specified in the enable parameter.

An enable value of true activates the channels to monitor setup and hold violations.

Examples:

```
See example for ILATrigger.IsMagniVuReadOnly.
```

2.34.2.6 ILATrigger.GetSetupHoldViolationEnable

Description:

Returns an array of boolean values that indicate which channels are monitoring setup and hold violations.

Declaration Syntax:

Visual Basic

```
Sub
GetSetupHoldViolationEnable ( channels As TlaCommon.IChannelBit )
                                                      As Boolean()
```

C#

```
bool[]      GetSetupHoldViolation( TlaCommon.IChannelBit[] channels )
```

C++

```
bool GetSetupHoldViolation ([in] IChannelBit* channels[] ) []
```

Arguments:

`channels` – an array of IChannelBits.

Return Value:

An array of boolean values indicating the monitoring state of the corresponding channel in the channels[] parameter.

Exceptions Thrown:

ArgumentException:
    Condition: Channel array is empty.
    Message: No channels specified. Array is empty.

ArgumentException:
    Condition: Illegal IChannelBit in the channels array.
    Message: Specified channel is not in the current instrument .

Remarks:

A value of true in the returned array indicates that the corresponding channel is monitoring setup and hold violations.

Examples:

```
See example for ILATrigger.IsMagniVuReadOnly.
```

2.34.3 ILATrigger Events

2.34.3.1 ILATrigger.MagniVuPositionChanged

Description:

When the MagniVu trigger position changes, the MagniVuPositionChanged event is raised.

Declaration Syntax:

Visual Basic

```
Event MagniVuPositionChanged As ObjectEventHandler
```

C#

```
event ObjectEventHandler MagniVuPositionChanged;
```

C++

```
__event  ObjectEventHandler MagniVuPositionChanged;
```

Event Data:

The ILATrigger object is passed as a parameter.

Remarks:

In addition to a caller's explicit alteration of the ILATriggerEvent.MagniVuPosition property, changes elsewhere in a data source's definition can modify the valid range of positions, which in turn, may cause an adjustment of the MagniVu trigger position. Any such change will raise the MagniVuPositionChanged event, regardless of the cause.

Examples:

```
See example for ILATrigger.IsMagniVuReadOnly.
```

2.34.3.2 ILATrigger.ForceMainPrefillChanged

Description:

When the force main memory prefill property changes, the ForceMainPrefillChanged event is raised.

Declaration Syntax:

Visual Basic

```
Event ForceMainPrefillChanged As ObjectEventHandler
```

C#

```
event ObjectEventHandler ForceMainPrefillChanged;
```

C++

```
__event  ObjectEventHandler ForceMainPrefillChanged;
```

Event Data:

The ILATrigger object is passed as a parameter.

Remarks:

Changes elsewhere in a the LA’s definition can override the main prefill storage behavior specified by the ForceMainPrefill property. Since such overrides do not change the value of the property, they do not raise this event.

Examples:

```
See example for ILATrigger.IsMagniVuReadOnly.
```

2.35 ILATriggerState

Description:

This is the IDataSourceTriggerState-based interface for all LA module trigger states. It contains only those members that must be implemented by all LA trigger modules.

Namespace: Tektronix.LogicAnalyzer.Common

Declaration Syntax:

Visual Basic

```
Interface ILATriggerState
        Inherits IDataSourceTriggerState
```

C#

```
interface ILATriggerState : IDataSourceTriggerState
```

C++

```
__gc __interface ILATriggerState : public IDataSourceTriggerState
```

Remarks:

The purpose of this interface is to allow methods that deal with trigger state properties common to all logic analyzer modules.

2.35.1 ILATriggerState Properties

2.35.1.1 ILATriggerState.Label

Description:

The comment string used to label the state.

Declaration Syntax:

Visual Basic

```
Property Label As String
```

C#

```
String Label { set; get; }
```

C++

```
String* get_Label ()
Void set_Label( String* label );
```

Arguments:

None.

Exceptions Thrown:

None.

Remarks:

None.

Examples:

C# code:

```
//
// Examples of using ILATriggerState properties and methods.
//
public bool TestLATriggerState ( ILATrigger laTrigger )
{
        // Start off with a defaulted trigger.
        // Get the first state and clause objects.
        laTrigger.Default();
        ILATriggerState trigState = laTrigger.GetLAState(0);
        ILATriggerClause clause = trigState.GetLAClause(0);

    string label = trigState.Label;
    clause.Label = "Testing";
    int numClauses = trigState.NumberOfClauses;
    bool canAdd1 = trigState.CanAddClause();
    bool canAdd2 = trigState.CanAddClause( clause );
    trigState.InsertClause( 0, clause );
    trigState.RemoveClause(0);
    trigState.GetClause(0);

    // Register for ILATriggerState events.
    // See ILATrigger example for how to register for events.
    RegisterForLATriggerStateEvents();

    // This will generate an ILATriggerState.LabelChanged event.
    trigState.Label = "testing";

    // This will generate an ILATriggerState.NumberOfClausesChanged event.
    trigState.AddClause( clause );
}
```

2.35.2 ILATriggerState Methods

2.35.2.1 ILATriggerState.GetLAClause

Description:

Returns the specified LA trigger clause from the trigger state.

Declaration Syntax:

Visual Basic

```
Function GetLAClause ( index As Integer ) As ILATriggerClause
```

C#

```
ILATriggerClause GetLAClause ( int index )
```

C++

```
ILATriggerClause * GetLAClause ( int index )
```

Arguments:

`index` – zero-based index of the LA trigger clause to return.

Return Value:

The interface to the specified LA trigger clause.

Exceptions Thrown:

ArgumentOutOfRangeException:
    Condition: Out-of-bounds clause index.
    Message: The clause index does not represent a valid trigger clause location.

Remarks:

This is a thin wrapper for casting the return value of IDataSourceTrigger.GetClause() to the appropriate LA trigger clause interface.

Examples:

```
See example for ILATriggerState.Label.
```

2.35.3 ILATriggerState Events

2.35.3.1 ILATriggerState.LabelChanged

Description:

When the LA trigger state’s Label property changes, the LabelChanged event is raised.

Declaration Syntax:

Visual Basic

```
Event LabelChanged As ObjectEventHandler
```

C#

```
event ObjectEventHandler LabelChanged;
```

C++

```
__event  ObjectEventHandler LabelChanged;
```

Event Data:

The ILATriggerState object is passed as a parameter.

Remarks:

None.

Examples:

```
See example for ILATriggerState.Label.
```

2.36 ILATriggerClause

Description:

This is the IDataSourceTriggerClause-based interface for all LA module trigger clauses. It contains only those members that must be implemented by all LA modules.

Namespace: Tektronix.LogicAnalyzer.Common

Declaration Syntax:

Visual Basic

```
Interface ILATriggerClause
      Inherits IDataSourceTriggerClause
```

C#

```
interface ILATriggerClause : IDataSourceTriggerClause
```

C++

```
__gc __interface ILATriggerClause : public IDataSourceTriggerClause
```

Remarks:

The purpose of this interface is to allow methods that deal with trigger clause properties common to all logic analyzer modules.

2.36.1 ILATriggerClause Properties

2.36.1.1 ILATriggerClause.Label

Description:

The comment string used to label the clause.

Declaration Syntax:

Visual Basic

```
Property Label As String
```

C#

```
String Label { set; get; }
```

C++

```
String* get_Label ()
Void set_Label( String* label );
```

Arguments:

`label` – string containing the comment label for the clause.

Exceptions Thrown:

None.

Remarks:

None.

Examples:

C# code:

```
//
// Examples of using ILATriggerClause properties and methods.
//
public bool TestLATriggerClause ( ILATrigger laTrigger )
{
        laTrigger.Default();

        ILATriggerClause clause = laTrigger.GetLAState(0).GetLAClause(0);
        ILATriggerEvent trigEvent = clause.GetLAEvent( 0 );
        ILATriggerAction trigAction = clause.GetLAAction( 0  );

        string label = clause.Label;
    string eventDef = trigEvent.Definition;
        string actionDef = trigAction.Definition;
    int numEvents = clause.NumberOfEvents;
    int actions = clause.NumberOfActions;
    bool canAddE1 = clause.CanAddEvent();
    bool canAddE2 = clause.CanAddEvent( trigEvent );
    clause.InsertEvent( 0, trigEvent );
    ILATriggerEvent getEvent = clause.GetLAEvent(0);
    clause.RemoveEvent( 0 );
    bool canAddA1 = clause.CanAddAction();
    bool canAddA2 = clause.CanAddAction( trigAction );
    clause.InsertAction( 0, trigAction );
    clause.RemoveAction(0);
    ILATriggerAction getAction = clause.GetLAAction(idx);

        // Register for ILATriggerClause events.
    // See ILATrigger example for how to register for events.
        RegisterLATriggerClauseEvents();

        // Generate an ILATriggerClause.LabelChanged event.
        clause.Label = "testing";

        // Generate an ILATriggerClause.EventLogicChanged event.
        // Need at least two events before EventLogic can be changed.
        clause.AddEvent(trigEvent);
        clause.EventLogic = LogicOperator.LogicalAnd;

        // Generate an ILATriggerClause.NumberOfEventsChanged event.
        clause.AddEvent( trigEvent );

        // Generate an ILATriggerClause.NumberOfActionsChanged event.
        clause.AddAction( trigAction );
}
```

2.36.1.2 ILATriggerClause.EventLogic

Description:

The boolean logic to use when evaluating events in the clause during an acquisition.

Declaration Syntax:

Visual Basic

```
Property EventLogic As LogicOperator
```

C#

```
LogicOperator EventLogic { set; get; }
```

C++

```
LogicOperator get_ EventLogic ()
Void set_EventLogic ( LogicOperator logic );
```

Arguments:

`operator` – logic operator to use when combining trigger events.
.

Exceptions Thrown:

ArgumentException :
Condition: Setting a single-event clause to LogicalOr.
Message: Single-event clauses can only be set to LogicalAnd.

Remarks:

All events in a clause are combined with the same logic operator. LA trigger clauses don't support mixed boolean expressions.

Examples:

```
See example for ILATriggerClause.Label.
```

2.36.2 ILATriggerClause Methods

2.36.2.1 ILATriggerClause.GetLAEvent

Description:

Returns the specified LA trigger event from the LA trigger clause.

Declaration Syntax:

Visual Basic

```
Function GetLAEvent ( index As Integer )
      As ILATriggerEvent
```

C#

```
ILATriggerEvent GetLAEvent ( int index )
```

C++

```
ILATriggerEvent * GetLAEvent ( int index )
```

Arguments:

`index` – zero-based index of the LA trigger event to return.

Return Value:

Interface to an LA trigger event.

Exceptions Thrown:

ArgumentException :
    Condition: Invalid event index.
    Message: The event index does not represent a valid event location in the trigger clause.

Remarks:

This is a thin wrapper for casting the return value of IDataSourceTrigger.GetEvent() to the appropriate LA trigger event interface.

Examples:

```
See example for ILATriggerClause.Label.
```

2.36.2.2 ILATriggerClause.GetLAAction

Description:

Returns the specified LA trigger action from the LA trigger clause.

Declaration Syntax:

Visual Basic

```
Function GetLAAction ( index As Integer )
      As ILATriggerAction
```

C#

```
ILATriggerAction GetLAAction ( int index )
```

C++

```
ILATriggerAction * GetLAAction ( int index )
```

Arguments:

`index` – zero-based index of the LA trigger action to return.

Return Value:

Interface to an LA trigger action.

Exceptions Thrown:

ArgumentException :
    Condition: Invalid action index.
    Message: The action index does not represent a valid action location in the trigger clause.

Remarks:

This is a thin wrapper for casting the return value of IDataSourceTrigger.GetAction() to the appropriate LA trigger action interface.

Examples:

```
See example for ILATriggerClause.Label.
```

2.36.3 ILATriggerClause Events

2.36.3.1 ILATriggerClause.LabelChanged

Description:

When the LA trigger clause's Label property changes, the LabelChanged event is raised.

Declaration Syntax:

Visual Basic

```
Event LabelChanged As ObjectEventHandler
```

C#

```
event ObjectEventHandler LabelChanged;
```

C++

```
__event  ObjectEventHandler LabelChanged;
```

Event Data:

The ILATriggerClause object is passed as a parameter.

Remarks:

None.

Examples:

```
See example for ILATriggerClause.Label.
```

2.36.3.2 ILATriggerClause.EventLogicChanged

Description:

When the LA trigger clause's EventLogic property changes, the EventLogicChanged event is raised.

Declaration Syntax:

Visual Basic

```
Event EventLogicChanged As ObjectEventHandler
```

C#

```
event ObjectEventHandler EventLogicChanged;
```

C++

```
__event  ObjectEventHandler EventLogicChanged;
```

Event Data:

The ILATriggerClause object is passed as a parameter.

Remarks:

When deletion of a trigger event reduces the number of events in the clause to 1, the EventLogic property automatically resets to a value of LogicalAnd. If that represents an actual change to the property's value, the EventLogicChanged event is raised.

The event is also raised whenever a caller directly changes the property's value.

Examples:

```
See example for ILATriggerClause.Label.
```

2.37 ILATriggerStorage

Description:

This is the ILATriggerClause-based interface for all LA module trigger global storage clauses. It contains only those members that must be implemented by all LA modules.

Namespace: Tektronix.LogicAnalyzer.Common

Declaration Syntax:

Visual Basic

```
Interface ILATriggerStorage
      Inherits ILATriggerClause
```

C#

```
interface ILATriggerStorage: ILATriggerClause
```

C++

```
__gc __interface ILATriggerStorage: public ILATriggerClause
```

Remarks:

The purpose of this interface is to allow methods that deal with trigger storage clause properties common to all logic analyzer modules. This interface controls default, global storage behavior for the LA trigger during an acquisition.

LA trigger events can be accessed from and added to the trigger storage definition, with some resource restrictions. The events are used to control storage when the global storage mode is set to use conditional storage. (See below for details.)

LA trigger actions are not accessible through this interface.

Actions within the trigger definition's state clauses can override the global storage behavior defined by the ILATriggerStorage interface.

2.37.1 ILATriggerStorage Properties

2.37.1.1 ILATriggerStorage.Label

Description:

The comment string used to label the clause. Unsupported.

Declaration Syntax:

Visual Basic

```
Property Label As String
```

C#

```
String Label { set; get; }
```

C++

```
String* get_Label ()
Void set_Label( String* label );
```

Arguments:

`label` – string containing the comment label for the clause.

Exceptions Thrown:

NotSupportedException:
Condition: The Label property isn't supported for the trigger storage definition.
Message: Trigger storage does not have a label property.

Remarks:

This ILATriggerClause property should not be accessed.

Examples:

```
This property is unsupported.
```

2.37.1.2 ILATriggerStorage.GlobalStorage

Description:

The global default storage operation to perform during an acquisition.

Declaration Syntax:

Visual Basic

```
Property GlobalStorage As GlobalStorageType
```

C#

```
GlobalStorageType GlobalStorage { set; get; }
```

C++

```
GlobalStorageType get_GlobalStorage ()
Void set_GlobalStorage ( GlobalStorageType storeType );
```

Arguments:

`storeType` – default global storage behavior to use during an acquisition.

Exceptions Thrown:

TlaTriggerResourceException:
Condition: Transitional storage conflicts with change events in the trigger definitions.
Message: Transitional storage cannot be used when a change event is in use already elsewhere in the trigger definition.

Remarks:

Not all LA modules throw the exception. Check ILATriggerStorage.IsGlobalStorageTypeValid() to determine whether this operation can be performed without throwing an assert.

Examples:

C# code:

```
//
// Examples of using ILATriggerStorageproperties properties and methods.
//
public bool TestLATriggerStorage ( ILATrigger laTrigger )
{
        laTrigger.Default();
        ILATriggerStorage trigStorage = laTrigger.GetGlobalStorage();

        GlobalStorageType setval = GlobalStorageType.Conditional;
        trigStorage.GlobalStorage = setval;
        GlobalStorageType getval = trigStorage.GlobalStorage;

        trigStorage.GlobalStorage = GlobalStorageType.StartStop;
        StartStopStoreType setval = StartStopStoreType.StopStore;
        trigStorage.InitialStartStop = setval;
        StartStopStoreType getval = trigStorage.InitialStartStop;

        GlobalStorageType type = GlobalStorageType.Transitional;
        bool isValid = trigStorage.IsGlobalStorageTypeValid( type );

        ILATriggerGroupSet groups = trigStorage.GetTriggerGroupsForTransitional();
        bool canAdd1 = trigStorage.CanAddEvent();

        ILATriggerEvent trigEvent = trigStorage.GetLAEvent(0);
        bool canAdd2 = trigStorage.CanAddEvent(trigEvent);

    trigStorage.AddEvent( trigEvent );
    trigStorage.InsertEvent( trigEvent );
    trigStorage.RemoveEvent(0);
    ILATriggerEvent getEvent = trigStorage.GetLAEvent(0);

    ILATriggerAction trigAction = trigStorage.GetLAAction( 0 );
    trigStorage.AddAction( trigAction );
    trigStorage.InsertAction( 0, trigAction );

        // Register for ILATriggerStorage events.
    // See ILATrigger example for how to register for events.
    RegisterLATriggerStorageEvents();

    // Generate an ILATriggerStorage.StorageChanged event.
    trigStorage.GlobalStorage = GlobalStorageType.Conditional;

    // Generate an ILATriggerStorage.StartStopChanged event.
    trigStorage.GlobalStorage = GlobalStorageType.StartStop;
    trigStorage.InitialStartStop = StartStopStoreType.StopStore;
}
```

2.37.1.3 ILATriggerStorage.InitialStartStop

Description:

For StartStop default storage, indicates the initial storage state.

Declaration Syntax:

Visual Basic

```
Property InitialStartStop As StartStopStoreType
```

C#

```
StartStopStoreType InitialStartStop { set; get; }
```

C++

```
StartStopStoreType get_InitialStartStop ()
Void set_InitialStartStop ( StartStopStoreType storeType );
```

Arguments:

`storeType` – Start/stop store mode to use at the beginning of an acquisition.

Exceptions Thrown:

InvalidOperationException:
Condition: ILATriggerStorage.GlobalStorageType is not set to GlobalStorageType.StartStop.
Message: Access to ILATriggerStorage.InitialStartStop is only valid when ILATriggerStorage.GlobalStorageType is set to GlobalStorageType.StartStop.

Remarks:

Check the ILATriggerStorage.GlobalStorageType property to make sure it is set to GlobalStorageType.StartStop before attempting to access ILATriggerStorage.StartStopStoreType.

Examples:

```
See example for ILATriggerStorage.GlobalStorage.
```

2.37.2 ILATriggerStorage Methods

2.37.2.1 ILATriggerStorage.IsGlobalStorageTypeValid

Description:

Indicates whether the specified global storage is valid for the current trigger definition.

Declaration Syntax:

Visual Basic

```
Function IsGlobalStorageTypeValid ( storeType As GlobalStorageType )
      As Boolean
```

C#

```
bool IsGroupNameValid ( GlobalStorageType storeType )
```

C++

```
bool IsGroupNameValid ( GlobalStorageType storeType )
```

Arguments:

`storeType` – default global storage behavior to use during an acquisition.

Return Value:

Value of true indicates that the storage type can be used in the current trigger definition.

Exceptions Thrown:

None.

Remarks:

None.

Examples:

```
See example for ILATriggerStorage.GlobalStorage.
```

2.37.2.2 ILATriggerStorage.GetTriggerGroupsForTransitional

Description:

Returns the set of trigger groups that define channel group monitoring during transitional global storage.

Declaration Syntax:

Visual Basic

```
Function GetTriggerGroupsForTransitional () As ILATriggerGroupSet
```

C#

```
ILATriggerGroupSet GetTriggerGroupsForTransitional ()
```

C++

```
ILATriggerGroupSet *  GetTriggerGroupsForTransitional ()
```

Arguments:

None.

Return Value:

The interface object that provides access to all trigger groups used to define event monitoring for transitional storage.

Exceptions Thrown:

InvalidOperationException:
Condition: ILATriggerStorage.GlobalStorageType is not set to GlobalStorageType.Transitional.
Message: Access to the trigger group set for transitional storage is only valid when ILATriggerStorage.GlobalStorageType is set to GlobalStorageType.Transitional.

Remarks:

All references to the trigger group set used for transitional storage become invalid when ILATriggerStorage.GlobalStorageType is no longer set to GlobalStorageType.Transitional.

Examples:

```
See example for ILATriggerStorage.GlobalStorage.
```

2.37.2.3 ILATriggerStorage.AddEvent

Description:

Overrides IDataSourceTriggerClause.AddEvent method.

Declaration Syntax:

Visual Basic

```
Sub AddEvent ( event as IDataSourceTriggerEvent )
```

C#

```
void AddEvent ( IDataSourceTriggerEvent event )
```

C++

```
void AddEvent ( IDataSourceTriggerEvent * event )
```

Arguments:

`event` – the pre-initialized event to be added.

Return Value:

None.

Exceptions Thrown:

In addition to the exceptions thrown by the base class methods:

InvalidOperationException:
- *Condition*: ILATriggerStorage.GlobalStorage property isn't set to GlobalStorageType.Conditional.
- *Message*: Global storage trigger events are undefined when ILATriggerStorage.GlobalStorage is not set to GlobalStorageType.Conditional.

Remarks:

None.

Examples:

```
See example for ILATriggerStorage.GlobalStorage.
```

2.37.2.4 ILATriggerStorage.InsertEvent

Description:

Overrides IDataSourceTriggerClause.InsertEvent method.

Declaration Syntax:

Visual Basic

```
Sub InsertEvent ( index As Integer,
                  event as IDataSourceTriggerEvent )
```

C#

```
void InsertEvent ( int index,
                   IDataSourceTriggerEvent event )
```

C++

```
void InsertEvent ( int index,
                   IDataSourceTriggerEvent * event )
```

Arguments:

`index` – zero-based index of the location in the trigger clause at which to add the new event.

`event` – the pre-initialized event to be added.

Return Value:

None.

Exceptions Thrown:

In addition to the exceptions thrown by the base class methods:

InvalidOperationException:
Condition: ILATriggerStorage.GlobalStorage property isn't set to GlobalStorageType.Conditional.
Message: Global storage trigger events are undefined when ILATriggerStorage.GlobalStorage is not set to GlobalStorageType.Conditional.

Remarks:

None.

Examples:

`See example for ILATriggerStorage.GlobalStorage.`

2.37.2.5 ILATriggerStorage.RemoveEvent

Description:

Overrides IDataSourceTriggerClause.RemoveEvent method.

Declaration Syntax:

Visual Basic

```
Sub RemoveEvent ( index As Integer )
```

C#

```
void RemoveEvent ( int index )
```

C++

```
void RemoveEvent ( int index )
```

Arguments:

`index` – zero-based index of the event to remove.

Return Value:

None.

Exceptions Thrown:

In addition to the exceptions thrown by the base class methods:

InvalidOperationException:
Condition: ILATriggerStorage.GlobalStorage property isn't set to GlobalStorageType.Conditional.
Message: Global storage trigger events are undefined when ILATriggerStorage.GlobalStorage is not set to GlobalStorageType.Conditional.

Remarks:

None.

Examples:

```
See example for ILATriggerStorage.GlobalStorage.
```

2.37.2.6 ILATriggerStorage.GetEvent

Description:

Overrides IDataSourceTriggerClause.GetEvent method.

Declaration Syntax:

Visual Basic

```
Function GetEvent ( index As Integer )
      As IDataSourceTriggerEvent
```

C#

```
IDataSourceTriggerEvent GetEvent ( int index )
```

C++

```
IDataSourceTriggerEvent * GetEvent ( int index )
```

Arguments:

index – zero-based index of the trigger event to return.

Return Value:

The interface to the specified trigger event.

Exceptions Thrown:

In addition to the exceptions thrown by the base class methods:

InvalidOperationException:
Condition: ILATriggerStorage.GlobalStorage property isn't set to GlobalStorageType.Conditional.
Message: Global storage trigger events are undefined when ILATriggerStorage.GlobalStorage is not set to GlobalStorageType.Conditional.

Remarks:

None.

Examples:

```
See example for ILATriggerStorage.GlobalStorage.
```

2.37.2.7 ILATriggerStorage.GetLAAction

Description:

Returns the specified LA trigger action from the LA trigger storage clause. Unsupported

Declaration Syntax:

Visual Basic

```
Function GetLAAction ( index As Integer )
      As ILATriggerAction
```

C#

```
ILATriggerAction GetLAAction ( int index )
```

C++

```
ILATriggerAction * GetLAAction ( int index )
```

Arguments:

`index` – zero-based index of the LA trigger action to return.

Return Value:

Undefined.

Exceptions Thrown:

NotSupportedException:
Condition: Access to actions isn't supported for the trigger storage definition.
Message: Trigger storage does not support trigger actions.

Remarks:

This ILATriggerClause method should not be called.

Examples:

```
This method is unsupported.
```

2.37.2.8 ILATriggerStorage.CanAddAction

Description:

Overrides IDataSourceTriggerClause.AddEvent method.

Declaration Syntax:

Visual Basic

```
Function CanAddAction () As Boolean
Function CanAddAction ( action As IDataSourceTriggerAction ) As Boolean
```

C#

```
bool CanAddAction ()
bool CanAddAction ( IDataSourceTriggerAction action )
```

C++

```
bool CanAddAction ()
bool CanAddAction ( IDataSourceTriggerAction * action )
```

Arguments:

`action` – a pre-defined trigger action.

Return Value:

Always returns false.

Exceptions Thrown:

None.

Remarks:

ILATriggerStorage does not support direct modification of trigger storage actions.

Examples:

```
See example for ILATriggerStorage.GlobalStorage.
```

2.37.2.9 ILATriggerStorage.AddAction

Description:

Appends a new action to the trigger clause. Unsupported

Declaration Syntax:

Visual Basic

```
Sub AddAction ( action as IDataSourceTriggerAction )
```

C#

```
void AddAction ( IDataSourceTriggerAction action )
```

C++

```
void AddAction ( IDataSourceTriggerAction * action )
```

Arguments:

`action` – the pre-initialized action to be added.

Return Value:

None.

Exceptions Thrown:

NotSupportedException:
Condition: Access to actions isn't supported for the trigger storage definition.
Message: Trigger storage does not support trigger actions.

Remarks:

This IDataSourceTriggerClause method should not be called.

Examples:

```
This method is unsupported.
```

2.37.2.10 ILATriggerStorage.InsertAction

Description:

Inserts a new action into the trigger clause at the specified location. Unsupported

Declaration Syntax:

Visual Basic

```
Sub AddActionAfter ( index As Integer,
                     action as IDataSourceTriggerAction )
```

C#

```
void AddActionAfter ( int index,
                      IDataSourceTriggerAction action )
```

C++

```
void AddActionAfter ( int index,
                      IDataSourceTriggerAction * action )
```

Arguments:

`index` – zero-based index of the location in the trigger clause at which to insert the new action.

`action` – the pre-initialized action to be added.

Return Value:

None.

Exceptions Thrown:

NotSupportedException:
Condition: Access to actions isn't supported for the trigger storage definition.
Message: Trigger storage does not support trigger actions.

Remarks:

This IDataSourceTriggerClause method should not be called.

Examples:

```
This method is unsupported.
```

2.37.2.11 ILATriggerStorage.RemoveAction

Description:

Removes the specified action from the trigger clause. Unsupported

Declaration Syntax:

Visual Basic

```
Sub RemoveAction ( index As Integer )
```

C#

```
void RemoveAction ( int index )
```

C++

```
void RemoveAction ( int index )
```

Arguments:

`index` – zero-based index of the action to remove.

Return Value:

None.

Exceptions Thrown:

NotSupportedException:
Condition: Access to actions isn’t supported for the trigger storage definition.
Message: Trigger storage does not support trigger actions..

Remarks:

This IDataSourceTriggerClause method should not be called.

Examples:

```
This method is unsupported.
```

2.37.2.12 ILATriggerStorage.GetAction

Description:

Returns the specified action from the trigger clause. Unsupported

Declaration Syntax:

Visual Basic

```
Function GetAction ( index As Integer )
      As IDataSourceTriggerAction
```

C#

```
IDataSourceTriggerAction GetAction ( int index )
```

C++

```
IDataSourceTriggerAction * GetAction ( int index )
```

Arguments:

`index` – zero-based index of the trigger action to return.

Return Value:

Undefined.

Exceptions Thrown:

NotSupportedException:
  Condition: Access to actions isn't supported for the trigger storage definition.
  Message: Trigger storage does not support trigger actions.

Remarks:

This IDataSourceTriggerClause method should not be called.

Examples:

```
This method is unsupported.
```

2.37.3 ILATriggerStorage Events

2.37.3.1 ILATriggerStorage.GlobalStorageChanged

Description:

When the LA trigger’s GlobalStorage property changes, the GlobalStorageChanged event is raised.

Declaration Syntax:

Visual Basic

```
Event GlobalStorageChanged As ObjectEventHandler
```

C#

```
event ObjectEventHandler GlobalStorageChanged;
```

C++

```
__event  ObjectEventHandler GlobalStorageChanged;
```

Event Data:

The ILATriggerStorage object is passed as a parameter.

Remarks:

None.

Examples:

```
See example for ILATriggerStorage.GlobalStorage.
```

2.37.3.2 ILATriggerStorage.InitialStartStopChanged

Description:

When the LA trigger storage's InitialStartStop property changes, the InitialStartStopChanged event is raised.

Declaration Syntax:

Visual Basic

```
Event InitialStartStopChanged As ObjectEventHandler
```

C#

```
event ObjectEventHandler InitialStartStopChanged;
```

C++

```
__event  ObjectEventHandler InitialStartStopChanged;
```

Event Data:

The ILATriggerStorage object is passed as a parameter.

Remarks:

None.

Examples:

```
See example for ILATriggerStorage.GlobalStorage.
```

2.38 ILATriggerEvent

Description:

This is the IDataSourceTriggerEvent-based interface for all LA trigger events. It contains only those members that must be implemented by all LA modules.

Namespace: Tektronix.LogicAnalyzer.Common

Declaration Syntax:

Visual Basic

```
Interface ILATriggerEvent
Inherits IDataSourceTriggerEvent
```

C#

```
interface ILATriggerEvent  : IDataSourceTriggerEvent
```

C++

```
__gc __interface ILATriggerEvent : public IDataSourceTriggerEvent
```

Remarks:

The purpose of this interface is to allow methods that deal with trigger event properties common to all LA modules.

2.38.1 ILATriggerEvent Properties

2.38.1.1 ILATriggerEvent.IsMultiGroupEvent

Description:

Indicates whether the event involves more than one channel group.

Declaration Syntax:

Visual Basic

```
ReadOnly Property IsMultiGroupEvent As Boolean
```

C#

```
bool IsMultiGroupEvent { get; }
```

C++

```
bool get_IsMultiGroupEvent ()
```

Arguments:

None.

Exceptions Thrown:

None.

Remarks:

Value of true indicates that the event monitors more than one channel group. It also means that a call to ILATriggerEvent.GetGroupSet() will return the interface to the set of trigger groups responsible for defining the multi-group event without throwing an exception.

Examples:

C# code:

```
//
// Examples of using ILATriggerEvent properties and methods.
//
public bool TestLATriggerEvent ( ILATrigger laTrigger )
{
        laTrigger.Default();

    ILATriggerClause clause = laTrigger.GetLAState(0).GetLAClause(0);
        ILATriggerEvent trigEvent = clause.GetLAEvent(0);

    bool isMulti = trigEvent.IsMultiGroupEvent;
    string getDef = trigEvent.Definition;
    trigEvent.Definition = "Group \"A3\" = Hex AB";
    bool isExprValid = trigEvent.IsExpressionValid( "Group \"A3\" = Hex AB" );
```

```
        // To get a MultiGroup event, we need to set
        // the trigger event definition to one of the
        // multi-group event types.
        string[] eventDefs =
        {
                        "Word = \"Word 1\"",
                        "Glitch",
                        "SetupHold",
                        "Transition \"Transition 1\" Occurs",
        };

        for ( int idx = 0; idx < eventDefs.Length; idx++ )
        {
                trigEvent.Definition = eventDefs[idx];
                bool isMulti = trigEvent.IsMultiGroupEvent;

                if ( isMulti )
                {
                        ILATriggerGroupSet groupSet = trigEvent.GetGroupSet();
                        // use groupSet …
                }
                else
                {
                        // handle error …
                }
        }

        // Register for ILATriggerEvent events.
    // See ILATrigger example for how to register for events.
        RegisterLATriggerEventEvents();

        // Generate an ILATriggerEvent.DefinitionChanged event.
        trigEvent.Definition = "Group \"A3\" = Hex AB";
}
```

2.38.1.2 ILATriggerEvent.Definition

Description:

The expression string representing the LA trigger event's definition.

Declaration Syntax:

Visual Basic

```
Property Definition As String
```

C#

```
String Definition { set; get; }
```

C++

```
void set_Definition ( String* expression )
String* get_Definition ()
```

Arguments:

`expression` – string that defines a LA trigger event.

Exceptions Thrown:

TlaTriggerResourceException:
Condition: Invalid expression string.
Message: The expression does not represent a valid trigger event definition.

TlaTriggerResourceException:
Condition: The expression causes event usage to exceed what trigger definition can support.
Message: The event cannot be used because it requires resources currently unavailable.

TlaTriggerResourceException:
Condition: Symbol referenced in the expression isn't in the specified symbol file.
Message: The expression's symbol is not present in the specified symbol file.

TlaFileOperationException :
Condition: The symbol file is not valid. When this is thrown the Failure member will be set to `InvalidFile`.
Message: The symbol file does not have a valid or recognizable format.

ArgumentException :
Condition: The `fileName` argument is not a valid filename.
Message: Invalid file name passed in the event definition.

PathTooLongException :
Condition: The file path contained in `filename` is too long.
Message: The passed-in file name is too long.

FileNotFoundException :
    Condition: Specified symbol file couldn't be found.
    Message: Symbol file not found.

FileLoadException :
    Condition: File was found, but can't be loaded.
    Message: Symbol file cannot be loaded.

Remarks:

This implementation of the base class' property generates several additional exceptions.

Query ILATriggerEvent.IsExpressionValid () to determine whether the Definition property can be set to the specified string without throwing an exception.

The following is a Backus-Naur Form description of LA trigger event syntax:

```
<Definition> ::= <WordEvent> |
                 <GroupEvent> |
                 <ChannelEvent> |
                 <TransitionEvent> |
                 <GlitchEvent> |
                 <SetupHoldEvent> |
                 <CounterEvent> |
                 <TimerEvent> |
                 <SignalEvent> |
                 <SnapshotEvent> |
                 Anything |
                 Nothing

<WordEvent> ::= Word <CompareOp> <WordDefName>

<GroupEvent ::= Group <GroupName> <CompareOp> <Radix> <WordValue> |
                Group <GroupName> Changes |
Group <GroupName> <RangeOpSingle> Radix <RangeValueSingle> |
Group <GroupName> <RangeOpDouble> Radix <RangeValueDouble>

<ChannelEvent> ::= Channel <ChannelName> <ChannelOp> <ChannelValue> |
                   Channel <ChannelName> <ChangeOp> <ChangeValue>

TransitionEvent ::= Transition <TransitionDefName> <TransitionOp>

SnapshotEvent ::= Snapshot <CompareOp> "Current Sample" |
                  Snapshot <LoadedOp>

GlitchEvent ::= Glitch

SetupHoldEvent ::= SetupHold

CounterEvent ::= Counter <CtrTmrNumber> <CtrTmrOp> <CtrValue>

TimerEvent ::= Timer <CtrTmrNumber> <CtrTmrOp> <TmrValue>

SignalEvent ::= Signal <SignalNumber> <BooleanValue>
```

```
<WordName> ::= "<AlphaNumericString>"
<TransitionName> ::= "<AlphaNumericString>"
<SnapshotName> ::= "<AlphaNumericString>"
<ChannelName> ::= "<AlphaNumericString>"
   NOTE: All whitespace characters other than the space character are
   invalid for <WordName>, < ChannelName >,
   <TransitionName>, and <SnapshotName>.

<GroupName> ::= "<AlphaNumericString>"
   NOTE: White space, other than a space character, and the single
   backquote (`) are invalid.

<WordOp> ::=  = | !=

<Radix> ::= Bin |
            Oct |
            Hex |
            Decimal |
            SignedDecimal |
            Symbolic <SymFile>

<SymFile> ::= "<Pathname>"

<WordValue> ::= <LiteralValue> |
                <SignedInteger>
                <SymbolName>

NOTE: A symbol name is valid only if the event's radix is set to Sym,
with a valid symbol file path specified. A signed integer value is
valid only if the event's radix is set to SignedDecimal.

<LiteralValue> ::=  <NumericValue> |
                    x | X | $
                    <LiteralValue><NumericValue> |
                    <LiteralValue>x |
                    <LiteralValue>X |
                    <LiteralValue>$ |

NOTE: The $ indicates a value containing don't cared bits that cannot
be represented in the current radix. The $ is for read back only, and
not to be used for setting an event value.

<NumericValue> ::= <Digit> |
                   <NumericValue><Digit>

<SignedInteger> ::= <DecimalValue> |
                    <Sign><DecimalValue>

<DecimalValue> ::= <DecimalDigit> |
                  <DecimalValue><DecimalDigit>

<Sign> ::= + | -

<Digit> ::= <DecimalDigit> |
            <HexadecimalDigit>

<DecimalDigit> ::= 0-9
```

```
<HexidecimalDigit> ::= a-f | A-F

<SymbolName> ::= "<AlphaNumericString>"

<ChannelOp> ::= =

<ChangeOp> ::= Goes |
               DoesntGo

<TransitionOp> ::= Occurs |
                   DoesntOccur

<CompareOp> ::=   = | !=

<LoadedOp> ::= IsLoaded |
               NotLoaded

<RangeOpSingle> ::= > | < | >= | <=

<RangeOpDouble> ::= IsIn | IsNotIn

<RangeValueSingle> ::=  <NumericValue> |
                        <SignedInteger> |
                        <SymbolName>

<RangeValueDouble> ::=  <RangeValueSingle> <RangeValueSingle>
   NOTE: Symbol files used for range events must be of range type.

<BooleanValue ::= True | False

<ChannelValue> ::= High | Low

<ChangeValue> ::= <ChannelValue> |
                  HighOrLow

<CtrTmrNumber> ::= 1 | 2

<CtrTmrOp> ::=  > | <=

<CtrValue> ::=  <SignedInteger>

<TmrValue> ::= <DecimalValue><TimePeriod>

<TimePeriod> ::= ns | ms | us | s

<SignalNumber> ::=      1 | 2 | 3 | 4
```

Examples:

C# code:

Here are some sample definitions. Also see example for ILATriggerEvent.IsMultiGroupEvent.

```
string[] eventDefs =
{
   "Word = \"Word 1\"",
   "Group \"A3\" = Hex AB",
   "Group \"A3\" Changes",
   "Group \"A3\" > Hex 12",
   "Group \"A3\" IsIn Hex 12 34",
   "Channel \"A0(0)\" = High",
   "Channel \"A0(0)\" Goes HighOrLow",
   "Transition \"Transition 1\" Occurs",
   "Glitch",
   "SetupHold",
   "Counter 1 <= 100",
   "Timer 2 > 124ns",
   "Signal 3 False",
   "Snapshot != \"Current Sample\"",
   "Snapshot IsNot \"Current Sample\"",
   "Snapshot Loaded",
   "Snapshot IsLoaded",
   "Anything",
   "Nothing"
};
```

2.38.2 ILATriggerEvent Methods

2.38.2.1 ILATriggerEvent.GetGroupSet

Description:

Returns the set of LA trigger groups used to define a multi-group LA trigger event.

Declaration Syntax:

Visual Basic

```
Function GetGroupSet () As ILATriggerGroupSet
```

C#

```
ILATriggerGroupSet GetGroupSet ()
```

C++

```
ILATriggerGroupSet * GetGroupSet ()
```

Arguments:

None.

Return Value:

The interface to a set of LA trigger groups.

Exceptions Thrown:

InvalidOperationException :
Condition: Event isn't multi-group.
Message: The event does not have a trigger group set.

Remarks:

Check ILATriggerEvent.IsMultiGroupEvent() to determine whether the operation can be performed without throwing an exception.

Examples:

```
See example for ILATriggerGroupSet.
```

2.39 ILATriggerAction

Description:

This is the IDataSourceTriggerAction-based interface for all LA trigger actions. It contains only those members that must be implemented by all LA trigger modules.

Namespace: Tektronix.LogicAnalyzer.Common

Declaration Syntax:

Visual Basic

```
Interface ILATriggerAction
      Inherits DataSourceTriggerAction
```

C#

```
abstract interface ILATriggerAction : IDataSourceTriggerAction
```

C++

```
__abstract __gc __interface ILATriggerAction
      : public IDataSourceTriggerAction
```

Remarks:

The purpose of this interface is to allow methods that deal with trigger action properties common to LA modules.

2.39.1 ILATriggerAction Properties

2.39.1.1 ILATriggerAction.Definition

Description:

The expression string representing the LA trigger action's definition.

Declaration Syntax:

Visual Basic

```
Property Definition As String
```

C#

```
String Definition { set; get; }
```

C++

```
void set_Definition ( String* expression )
String* get_Definition ()
```

Arguments:

`expression` – string that defines a LA trigger action.

Exceptions Thrown:

ArgumentException:
 Condition: Invalid expression string.
 Message: The expression does not represent a valid trigger action definition.

Remarks:

This implementation of the base class' property generates several additional exceptions.

Check ILATriggerAction.IsExpressionValid () to determine whether this operation can be performed without throwing an exception.

The following is a Backus-Naur Form description of LA trigger action syntax:

```
<Definition> ::= <TriggerAction |
                 <CounterAction> |
                 <TimerAction> |
                 <SignalAction> |
                 <GoToAction> |
                 <StoreAction> |
                 <SnapshotAction>
                 DoNothing

<TriggerAction> ::= TriggerAll |
                    TriggerSelf |
                    TriggerMain |
                    TriggerMagniVu

<CounterAction> ::= <CounterOp> <CtrTmrNumber>

<TimerAction> ::= <TimerOp> <CtrTmrNumber>

<SignalAction> ::= <SignalOp> <SignalNumber>

<GoToAction> ::= GoTo <StateNumber>

<StoreAction> ::= StoreSample |
                  DontStore |
                  StartStoring |
                  StopStoring

<SnapshotAction> ::= SnapshotSample


<CounterOp> ::= IncCounter |
                DecCounter |
                ResetCounter

<TimerOp> ::= StartTimer |
              ClearTimer |
              StopTimer

<CtrTmrNumber> ::=    1 | 2

<SignalOp> ::= SetSignal |
               ClearSignal

<SignalNumber> ::= 1 | 2 | 3 | 4

<StateNumber> ::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12
```

Examples:

C# code:

```
//
// Examples of using ILATriggerAction properties and methods.
//
public bool TestLATriggerAction( ILATrigger laTrigger )
{
        laTrigger.Default();

        ILATriggerClause clause = laTrigger.GetLAState( 0 ).GetLAClause( 0 );
        ILATriggerAction trigAction = clause.GetLAAction( 0 );

    trigAction.Definition = "TriggerSelf";
        string getDef = trigAction.Definition;

        bool isExprValid = trigAction.IsExpressionValid("IncCounter 1");

        // Register for ILATriggerAction events.
    // See ILATrigger example for how to register for events.
    RegisterLATriggerActionEvents();

    // Generate an ILATriggerAction.DefinitionChanged event.
        trigAction.Definition = "DoNothing";
}
```

Also, here are some sample action definitions:

```
string[] actionDefs =
{
    "TriggerAll",
    "TriggerSelf",
    "TriggerMain",
    "TriggerMagniVu",
    "IncCounter 1",
    "DecCounter 2",
    "ResetCounter 1",
    "StartTimer 1",
    "ClearTimer 2",
    "StopTimer 1",
    "SetSignal 1",
    "ClearSignal 2",
    "GoTo 2",
    "StoreSample",
    "DontStore",
    "StartStoring",
    "StopStoring",
    "SnapshotSample",
    "DoNothing"
};
```

2.40 ILATriggerGroupSet

Description:

This is the base interface for all collections of LA trigger groups that define a multi-group trigger event. It contains only those members that must be implemented by all LA modules.

Namespace: Tektronix.LogicAnalyzer.Common

Declaration Syntax:

Visual Basic

```
Interface ILATriggerGroupSet
        Inherits IValidity
        Inherits IDefault
        Inherits IDisposable
        Inherits ICloneable
```

C#

```
interface ILATriggerGroupSet
      : IValidity, IDefault, IDisposable, ICloneable
```

C++

```
__gc __interface ILATriggerGroupSet
      : public IValidity, IDefault, IDisposable, ICloneable
```

Remarks:

The purpose of this interface is to allow methods that deal with trigger group sets common to all LA modules.

2.40.1 ILATriggerGroupSet Properties

2.40.1.1 ILATriggerGroupSet.NumberOfTriggerGroups

Description:

Indicates the number of LA trigger groups in the set.

Declaration Syntax:

Visual Basic

```
ReadOnly Property NumberOfTriggerGroups As Int32
```

C#

```
Int32 NumberOfTriggerGroups { get; }
```

C++

```
Int32 get_NumberOfTriggerGroups ()
```

Arguments:

None.

Exceptions Thrown:

None.

Remarks:

None.

Examples:

C# code:

```
//
// Examples of using ILATriggerGroupSet properties and methods.
// Note that only the Word group set is changeable.
//
public bool TestLATriggerGroupSet ( ILATrigger laTrigger )
{
        laTrigger.Default();

        ILATriggerClause clause = laTrigger.GetLAState(0).GetLAClause(0);
        ILATriggerEvent trigEvent = clause.GetLAEvent(0);

    trigEvent.Definition = "Word = \"Word 1\"";

        ILATriggerGroupSet trigGroupSet = trigEvent.GetGroupSet();
    int numTrigGroups = trigGroupSet.NumberOfTriggerGroups;
```

```
        // Lool through all groups in trigGroupSet.
        for ( int idx = 0; idx < numTrigGroups; idx++ )
        {
                ILATriggerGroup trigGroup = trigGroupSet.GetTriggerGroup(idx);

                // process trigGroup …
        }

        // Register for ILATriggerGroupSet events.
    // See ILATrigger example for how to register for events.
        RegisterLATriggerGroupSetEvents();

        // Create a new group and add it to the current module.
        IChannelGrouping chanGrouping = laModule.GetChannelGroupingObject();
        IChannelGroup newGroup = chanGrouping.CreateGroup("Foo");

        // Generate an ILATriggerGroupSet.NumberOfGroupsChanged event.
        chanGrouping.AddGroup(newGroup);
}
```

2.40.2 ILATriggerGroupSet Methods

2.40.2.1 ILATriggerGroupSet.GetTriggerGroup

Description:

Returns the specified trigger group interface.

Declaration Syntax:

Visual Basic

```
Function GetTriggerGroup ( index As Int32 ) As ILATriggerGroup
```

C#

```
ILATriggerGroup GetTriggerGroup ( int index )
```

C++

```
ILATriggerGroup* GetTriggerGroup ( int index )
```

Arguments:

`index` – Zero-based index of an LA trigger group in the set.

Return Value:

The interface to the LA trigger group object.

Exceptions Thrown:

ArgumentException :
    Condition: Invalid trigger group index.
    Message: The trigger group index does not represent a valid location in the trigger group set.

Remarks:

Check ILATriggerGroupSet.NumberOfTriggerGroups() to determine the range of valid indices.

Examples:

```
See example for ILATriggerGroupSet.NumberOfTriggerGroups.
```

2.40.3 ILATriggerGroupSet Events

2.40.3.1 ILATriggerGroupSet.NumberOfTriggerGroupsChanged

Description:

When the number of LA trigger groups defined changes, the NumberOfTriggerGroupsChanged event is raised.

Declaration Syntax:

Visual Basic

```
Event NumberOfTriggerGroupsChanged As ObjectEventHandler
```

C#

```
event ObjectEventHandler NumberOfTriggerGroupsChanged;
```

C++

```
__event  ObjectEventHandler NumberOfTriggerGroupsChanged;
```

Event Data:

The ILATriggerGroupsSet object is passed as a parameter.

Remarks:

Changes elsewhere in the LA module definition can alter the number of trigger groups defined for trigger event usage. This event is raised whenever such a change occurs.

Examples:

```
See example for ILATriggerGroupSet.NumberOfTriggerGroups.
```

2.41 ILATriggerGroup

Description:

This is the base interface for an LA trigger group used in a multi-group trigger events definition. It contains only those members that must be implemented by all LA modules.

Namespace: Tektronix.LogicAnalyzer.Common

Declaration Syntax:

Visual Basic

```
Interface ILATriggerGroup
      Inherits IValidity
      Inherits IDefault
      Inherits IDisposable
      Inherits ICloneable
```

C#

```
interface ILATriggerGroup
      : IValidity, IDefault, IDisposable, ICloneable
```

C++

```
__gc __interface ILATriggerGroup
      : public IValidity, IDefault, IDisposable, ICloneable
```

Remarks:

The purpose of this interface is to allow methods that deal with trigger group properties common to all LA modules. Used as elements in an ILATriggerGroupSet, trigger groups define group-based properties for multi-group trigger events.

2.41.1 ILATriggerGroup Properties

2.41.1.1 ILATriggerGroup.IsDefinitionReadOnly

Description:

Indicates whether the Definition property can only be read in the current multi-group event's context.

Declaration Syntax:

Visual Basic

```
ReadOnly Property IsDefinitionUsed As Boolean
```

C#

```
bool IsDefinitionUsed { get; }
```

C++

```
bool get_ IsDefinitionUsed ()
```

Arguments:

None.

Exceptions Thrown:

None.

Remarks:

Some types of multi-group events only require information about whether a group is monitored for the event. In such cases, the Definition property is read-only and only contains the group's name.

Examples:

C# code:

```
//
// Examples of using ILATriggerGroup properties and methods.
// Note that only the Word group set is changeable.
//
public bool TestLATriggerGroup ( ILATrigger laTrigger )
{
        laTrigger.Default();

        ILATriggerClause clause = laTrigger.GetLAState(0).GetLAClause(0);
        ILATriggerEvent trigEvent = clause.GetLAEvent(0);

        ITLATriggerGroupSet trigGroupSet = null;
    ILATriggerGroup trigGroup = null;
```

```
// The only kind of events allowed for LATriggerGroup.
// Only the Word event is read/write.
string[] eventDefs =
{
        "Word = \"Word 1\"",                                              // group defs R/W
        "Glitch",                                                                  // group
defs W-only
        "Snapshot Loaded",                                                   // group defs W-
only
        "Transition \"Transition 1\" Occurs",                    // group defs W-only
};

        // Test for all event types.
        for (int idx = 0; idx < eventDefs.Length; idx++)
        {
                trigEvent.Definition = eventDefs[idx];
                trigGroupSet = trigEvent.GetGroupSet();
                int numTrigGroups = trigGroupSet.NumberOfTriggerGroups;

                // Test for all groups in trigGroupSet.
                for ( int jdx = 0; jdx < numTrigGroups; jdx++ )
                {
                        // Get the current trigger group definition.
                        ILATriggerGroup trigGroup = trigGroupSet.GetTriggerGroup(jdx);
                        bool isReadOnly = trigGroup.IsDefinitionReadOnly;
                }
        }

        // Test using the following definition.
    trigEvent.Definition = "Word = \"Word 1\"";

        IChannelGrouping chanGrouping = laModule.GetChannelGroupingObject();
        ArrayList groups = chanGrouping.Groups;

        trigGroupSet = trigEvent.GetGroupSet();

        // Test for all default groups: CK0, A3, A2
        for ( int idx = 0; idx < groups.Count; idx++ )
        {
                IChannelGroup group = (IChannelGroup) groups[idx];
                string groupName = group.Name;

                // Get the current trigger group definition.
                trigGroup = trigGroupSet.GetTriggerGroup(idx);

                // Test some properties.
        bool canEnable = trigGroup.CanEnable;
                trigGroup.Enable = true;
        bool isValid =
            trigGroup.IsExpressionValid( "Group \"A3\" = Hex AB" );

                // All of the trigger groups should be read/write for a Word event.
                // Transition, Glitch, and Snapshot events are read-only.
                if ( ! trigGroup.IsDefinitionReadOnly )
                {
                        // Set the new trigger group definition.
                        string newDef = "Group \"" + groupName + "\"" + " = Hex AB";
                        trigGroup.Definition = newDef;
                }
                else
                {
                        // handle error …
                }
        }

        // Set up for generating events.
        trigEvent.Definition = "Word = \"Word 1\"";
        trigGroupSet = trigEvent.GetGroupSet();
        trigGroup = trigGroupSet.GetTriggerGroup(0);

        // Register for ILATriggerGroup events.
    // See ILATrigger example for how to register for events.
    RegisterLATriggerGroupEvents();

    // Generate an ILATriggerGroup.DefinitionChanged event.
        trigGroup.Definition = "Group \"A3\" = Hex AB";

    // Generate an ILATriggerGroup.EnableChanged event.
    trigGroup.Enable = false;
}
```

2.41.1.2 ILATriggerGroup.Definition

Description:

The expression string representing the settings associated with a group used in a particular multi-group trigger event.

Declaration Syntax:

Visual Basic

```
Property Definition As String
```

C#

```
String Definition { set; get; }
```

C++

```
void set_Definition ( String* expression )
String* get_Definition ()
```

Arguments:

`expression` – string that defines a single group event within a multi-group trigger event.

Exceptions Thrown:

InvalidOperationException :
Condition: Property is read-only for the current multi-group event.
Message: The current trigger event does not support writeable multi-group expressions.

NullArgumentException :
Condition: Expression parameter is null or empty.
Message: N/A.

TlaTriggerResourceException:
Condition: Invalid trigger group expression string.
Message: The expression does not represent a valid group-based event for the current multi-group trigger event.

ArgumentException :
Condition: Target value is invalid.
Message: The specified target value is invalid.

ArgumentException :
Condition: Event operator is invalid.
Message: The expression’s operator is invalid.

FileNotFoundException :
Condition: Specified symbol file couldn’t be found.
Message: Symbol file not found.

FileLoadException :

Condition: File was found, but can't be loaded.
Message: Symbol file cannot be loaded.

ArgumentException :
Condition: File isn't recognized as a TLA-compatible symbol file.
Message: File is not a TLA-compatible symbol file.

TlaTriggerResourceException:
Condition: Symbol referenced in the expression isn't in the specified symbol file.
Message: The expression's symbol is not present in the specified symbol file.

TlaTriggerResourceException:
Condition: The expression causes event usage to exceed what the trigger definition can support.
Message: The expression cannot be used because it requires resources currently unavailable.

Remarks:

Check ILATriggerGroup.IsValidExpression () to determine whether setting the property can be performed without throwing an exception.

The following is a Backus-Naur Form description of LA trigger group definition syntax:

```
<Definition> ::= <WordEventGroupDefinition> |
                 <TransitionEventGroupDefinition> |
                 <GlitchEventGroupDefinition> |
                 <SnapshotEventGroupDefinition>

<WordEventGroupDefinition> ::= <GroupName> = <Radix> <WordValue>

<TransitionEventGroupDefinition> ::= <GroupName>

<GlitchEventGroupDefinition> ::= <GroupName>

<SnapshotEventGroupDefinition>::= <GroupName>

<Radix> ::= Bin |
            Oct |
            Hex |
            Decimal |
            SignedDecimal |
            Symbolic <SymFile>

<SymFile> ::= "<Pathname>"

<WordValue> ::= <LiteralValue> |
                <SymbolName>

NOTE: A symbol name is valid only if the event's radix is set to Sym,
with a valid symbol file path specified. A signed integer value is
valid only if the event's radix is set to SignedDecimal.
```

```
<LiteralValue> ::=  <NumericValue> |
                    x | X | $
                    <LiteralValue><NumericValue> |
                    <LiteralValue>x |
                    <LiteralValue>X |
                    <LiteralValue>$ |

NOTE: The $ indicates a value containing don’t cared bits that cannot
be represented in the current radix. The $ is for read back only, and
not to be used for setting an event value.

<NumericValue> ::= <Digit> |
                   <NumericValue><Digit>

<SignedInteger> ::= <DecimalValue> |
                    <Sign><DecimalValue>

<DecimalValue> ::= <DecimalDigit> |
                  <DecimalValue><DecimalDigit>

<Sign> ::= + | -

<Digit> ::= <DecimalDigit> |
            <HexadecimalDigit>

<DecimalDigit> ::= 0-9

<HexidecimalDigit> ::= a-f | A-F

<SymbolName> ::= “<AlphaNumericString>”
```

Examples:

```
See example for ILATriggerGroup.IsDefinitionReadOnly.
Also, here are some definition examples:
```

```
string[] wordTriggerGroupExprs =
{
   "Group \"A3\" = Hex AB",
   "Group \"A3\" = Bin 1011",
   "Group \"A3\" = Oct 377",
   "Group \"A3\" = Decimal 57",
   "Group \"A3\" = SignedDecimal -25",
   "Group \"A3\" = Symbolic
      \"C:\\Program Files\\TLA 700\\Samples\\TLA700 Samples\\Symbol Files\\AsciSym.tsf\"
      \"ACK\"",
};
```

2.41.1.3 ILATriggerGroup.CanEnable

Description:

Indicates whether the trigger group can be enabled for monitoring in the current multi-group trigger event.

Declaration Syntax:

Visual Basic

```
ReadOnly Property CanEnable As Boolean
```

C#

```
bool CanEnable { get; }
```

C++

```
bool get_CanEnable ()
```

Arguments:

None.

Exceptions Thrown:

None.

Remarks:

Under certain circumstances, a multi-group trigger event’s groups cannot be enabled without violating the integrity of the overall trigger definition.

Examples:

```
See example for ILATriggerGroup.IsDefinitionReadOnly.
```

2.41.1.4 ILATriggerGroup.Enable

Description:

Indicates whether the trigger group is enabled for monitoring in the current multi-group trigger event.

Declaration Syntax:

Visual Basic

```
Property Enable As Boolean
```

C#

```
bool Enable { set; get; }
```

C++

```
void set_Enable ( bool enable )
bool get_Enable ()
```

Arguments:

`enable` – true indicates group is used in the event.

Exceptions Thrown:

TlaTriggerResourceException:
- *Condition*: Enabling the group causes event usage to exceed what trigger definition can support.
- *Message*: The group cannot be enabled because it requires resources currently unavailable.

Remarks:

Query the ILATriggerEvent.CanEnable property to determine whether the operation can be performed without throwing an exception.

Examples:

```
See example for ILATriggerGroup.IsDefinitionReadOnly.
```

2.41.2 ILATriggerGroup Methods

2.41.2.1 ILATriggerGroup.IsExpressionValid

Description:

Indicates whether the specified expression describes a valid group event within the current, multi-group event.

Declaration Syntax:

Visual Basic

```
Function IsExpressionValid ( expression As String )
      As Boolean
```

C#

```
bool IsExpressionValid ( String expression )
```

C++

```
bool IsExpressionValid ( String* expression )
```

Arguments:

`expression` – string expression that defines a single group event within a multi-group event.

Return Value:

Value of true indicates the following:

- If ILATriggerGroup.IsDefinitionReadOnly returns false.
- The specified expression represents a valid group event for the current multi-group LA trigger event.

Exceptions Thrown:

None.

Remarks:

Check this method to determine whether assigning the event to ILATriggerGroup.Definition will throw an exception.

Examples:

```
See example for ILATriggerGroup.IsDefinitionReadOnly.
```

2.41.3 ILATriggerGroup Events

2.41.3.1 ILATriggerGroup.DefinitionChanged

Description:

When a trigger group's Definition property changes, the DefinitionChanged event is raised.

Declaration Syntax:

Visual Basic

```
Event DefinitionChanged As ObjectEventHandler
```

C#

```
event ObjectEventHandler DefinitionChanged;
```

C++

```
__event  ObjectEventHandler DefinitionChanged;
```

Event Data:

The IDataSourceTriggerGroup object is passed as a parameter.

Remarks:

In addition to a caller's explicit alteration of the ILATriggerGroup.Definition property, changes elsewhere in a data source's definition can modify a trigger event's expression. Any such change will raise the DefinitionChanged event, regardless of the cause.

Examples:

```
See example for ILATriggerGroup.IsDefinitionReadOnly.
```

2.41.3.2 ILATriggerGroup.EnableChanged

Description:

When a trigger group’s Enable changes, the EnableChanged event is raised.

Declaration Syntax:

Visual Basic

```
Event EnableChanged As ObjectEventHandler
```

C#

```
event ObjectEventHandler EnableChanged;
```

C++

```
__event  ObjectEventHandler EnableChanged;
```

Event Data:

The IDataSourceTriggerGroup object is passed as a parameter.

Remarks:

In addition to a caller’s explicit alteration of the ILATriggerGroup.Enable property, changes elsewhere in a data source’s definition can modify a trigger event’s expression. Any such change will raise the ExpressionChanged event, regardless of the cause.

Examples:

```
See example for ILATriggerGroup.IsDefinitionReadOnly.
```

3 Common Interfaces

This section contains .NET interfaces that can be implemented both by the TLA application and by plug-ins.

3.1 IValidity

Description:

Objects that represent parts of the TLA system that can be become invalid implement this interface. IValidity provides a means to programmatically check whether an object is currently valid, and whether it has becoming permanently invalid. It notifications clients whenever an object changes its validity.

Namespace: Tektronix.LogicAnalyzer.Common

Declaration Syntax:

Visual Basic

```
Interface IValidity
```

C#

```
interface IValidity
```

C++

```
__gc __interface IValidity
```

Remarks:

Clients that are notified that an object has become permanently invalid should release all references to the object, whose contents may no longer be accessible.

For some objects, validity many come and go, in which case, clients must deal intelligently with both situations.

3.1.1 IValidity Properties

3.1.1.1 IValidity.IsValid

Description:

This Boolean property indicates whether the object that has implemented the IValidity interface is in a valid state. True indicates that it is. A change in the property's state raises the ValidityChanged event.

Declaration Syntax:

Visual Basic

```
Property IsValid As Boolean
```

C#

```
bool IsValid { set; get;}
```

C++

```
bool get_IsValid ();
void set_IsValid ( bool );
```

Arguments:

None.

Exceptions Thrown:

None.

Remarks:

If the IsGarbage property goes true, IsValid MUST be PERMANENTLY false, by definition.

3.1.1.2 IValidity.IsGarbage

Description:

This Boolean property indicates whether the object that has implemented the IValidity interface is now permanently invalid. True indicates that it is. When this property goes true, it raises the ValidityChanged event.

Declaration Syntax:

Visual Basic

```
Property IsGarbage As Boolean
```

C#

```
bool IsGarbage { set; get;}
```

C++

```
bool get_IsGarbage ();
void set_IsGarbage ( bool );
```

Arguments:

None.

Exceptions Thrown:

None.

Remarks:

If the IsGarbage property goes true, IsValid MUST be PERMANENTLY false, by definition.

3.1.2 IValidity Events

3.1.2.1 IValidity.ValidityChanged

Description:

An object that implements IValidity raises this event when the object has its IsValid property's state changed.

Declaration Syntax:

Visual Basic

```
Event Invalid As EventHandler
```

C#

```
event EventHandler Invalid;
```

C++

```
__event EventHandler Invalid;
```

Event Data:

None.

Remarks:

Delegates that handle this event should check both the IsValid property to determine what state the object is in. If IsValid is false, the IsGarbage property indicates whether the object is permanently invalid. All references to a permanently invalid object should be released.

3.2 IDefault

Description:

Objects that represent parts of the TLA system that can be defaulted implement this interface. IDefault provides a means to programmatically default parts of the system, and it allows notification of clients when a default occurs.

Namespace: Tektronix.LogicAnalyzer.Common

Declaration Syntax:

Visual Basic

```
Interface IDefault
```

C#

```
interface IDefault
```

C++

```
__gc __interface IDefault
```

Remarks:

Plug-ins that can be meaningfully defaulted should implement this interface. When the system is defaulted, it will default any persistent plug-ins that implement IDefault.

3.2.1 IDefault Methods

3.2.1.1 IDefault.Default

Description:

When an object implements IDefault, this method causes the object to return to its default state.

Declaration Syntax:

Visual Basic

```
Sub Default ()
```

C#

```
void Default ()
```

C++

```
void Default ()
```

Arguments:

None.

Return Value:

None.

Exceptions Thrown:

None.

Remarks:

The TLA implements this interface in objects that represent the system and its modules.

3.2.2 IDefault Events

3.2.2.1 IDefault.Defaulted

Description:

An object that implements IDefault raises this event when the object has been returned to a default state. Generally an object is defaulted either through the user interface or programmatically through its Default method.

Declaration Syntax:

Visual Basic

```
Event Defaulted As EventHandler
```

C#

```
event EventHandler Defaulted;
```

C++

```
__event EventHandler Defaulted;
```

Event Data:

None.

3.3 IExportData

Description:

This interface is implemented by objects that can export their data to a stand-alone file

Namespace: Tektronix.LogicAnalyzer.Common

Declaration Syntax:

Visual Basic

```
Interface IExportData
```

C#

```
interface IExportData
```

C++

```
__gc __interface IExportData
```

3.3.1 IExportData Methods

3.3.1.1 IExportData.Export

Description:

This method exports data from the object into to a specified file.

Declaration Syntax:

Visual Basic

```
Sub Export (file As String, args As Ojbect)
```

C#

```
void Export (string file, object args)
```

C++

```
void Export (String* file, Object* args)
```

Arguments:

`file` – The full path of the file that will contain the exported data.

`args` – An object that contains parameters that refine what data will be exported or how it should be exported. The actual type of object depends on the implementation. In general implementations should allow `args` to be null.

Return Value:

None.

Exceptions Thrown:

ArgumentNullException :
Condition: The file argument is null.
Message: Null argument provided to Export.

ArgumentException :
Condition: The file argument does not represent a valid path.
Message: The file name or path is invalid.

PathTooLongException :
Condition: The file path contained in `filename` is too long.
Message: The passed-in file name is too long.

Remarks:

Since many problems can occur during file operations, the Export method should always be called within a try block, so that exceptions can be caught. The above list of possible exceptions cannot be guaranteed complete, so a catchall clause can be used in addition to catching specific exceptions of interest.

3.4 IDataWindow

Description:

This is the base interface for all data windows, including Listing, Waveform, and plug-in data windows. IDataWindow contains only those members that must be implemented by all data windows that are accessible through TPI.

Namespace: Tektronix.LogicAnalyzer.Common

Declaration Syntax:

Visual Basic

```
Interface IDataWindow
      Inherits IValidity
```

C#

```
interface IDataWindow : IValidity
```

C++

```
__gc __interface IDataWindow : public IValidity
```

Remarks:

The purpose of this interface is to allow methods that deal with all kinds of data windows to reference them all with the same type, and also to allows differentiation of data windows from other kinds of objects held in a collection.

3.4.1 IDataWindow Properties

3.4.1.1 IDataWindow.UserName

Description:

The user name of the data window as it appears to users. For data windows that appear as icons in the system window, this is the name that appears with the icon.

Declaration Syntax:

Visual Basic

```
Property UserName As String
```

C#

```
string UserName { set; get;}
```

C++

```
String* get_UserName ();
void set_UserName (String*);
```

Arguments:

None.

Exceptions Thrown:

ArgumentNullException:
Condition: The property is set to a null reference.
Message: <The message string is implementation specific.>

ArgumentException:
Condition: The property is set name that is already used by another data source in the system.
Message: The given name is already in use by another data source.

Remarks:

Some implementations might allow the user name to be set to null without throwing an exception. Clients of this interface should assume that null user names are not allowed.

3.4.2 IDataWindow Methods

3.4.2.1 IDataWindow.GetDataSources

Description:

Gets an array of all the data sources currently used to display data in the data window.

Declaration Syntax:

Visual Basic

```
Function GetDataSources () As IDataSource()
```

C#

```
IDataSource[] GetDataSources ();
```

C++

```
IDataSource* GetDataSources () __gc[];
```

Arguments:

None

Return Value:

If the data window is using any data sources, then an array of references to those data sources is returned. If no data sources are being used, then return value is null.

Exceptions Thrown:

None.

3.4.3 IDataWindow Events

3.4.3.1 IDataWindow.UserNameChanged

Description:

Data windows fire this event when the UserName property of the object changes.

Declaration Syntax:

Visual Basic

```
Event UserNameChanged As EventHandler
```

C#

```
event EventHandler UserNameChanged;
```

C++

```
__event EventHandler UserNameChanged;
```

Event Data:

The UserName property of the data window will contain the new name.

Remarks:

The TLA requires that all data windows have unique names. It will subscribe to this event for every data window plug-in in the system. If a name change causes a duplicate name in the system, the TLA will set it to a unique name.

3.5 ILockingWindow

Description:

By implementing this interface, data window plug-ins indicate their ability to participate in the Lock Windows dialog. The application uses ILockingWindow to lock together the timestamp values of cursors and screen centers of multiple data windows. The intended consumer of this interface is the TLA application alone.

Namespace: Tektronix.LogicAnalyzer.PlugIns

Declaration Syntax:

Visual Basic

```
Interface ILockingWindow
      Inherits IDataWindow
```

C#

```
interface ILockingWindow : IDataWindow
```

C++

```
__gc __interface ILockingWindow : IDataWindow
```

Remarks:

Data window plug-ins that implement ILockingWindow can be locked to TLA Waveform and Listing windows as well as to other data window plug-ins that implement it.

The TLA application coordinates the locking of data window plug-ins by handling change events from windows changed due to user interaction. The TLA then updates the position of a set of locked windows by calling ILockingWindow methods on windows that need to be updated.

This interface assumes that any implementing object has two cursors and a center display position. Data window plug-ins should implement this interface only if they have meaningful representations of these screen concepts.

To avoid problems with excessive numbers of events, any object implements this interface (including implementing its derived interfaces) should not subscribe to the ItemChanged event of any ILockingWindow object.

3.5.1 ILockingWindow Properties

3.5.1.1 ILockingWindow.ActiveCursor

Description:

Gets or sets which cursor is active within a locking data window

Declaration Syntax:

Visual Basic

```
Property ActiveCursor As LockingWindowItem
```

C#

```
LockingWindowItem ActiveCursor { set; get; }
```

C++

```
LockingWindowItem get_Property-Name ();
void set_Property-Name (LockingWindowItem);
```

Arguments:

None.

Exceptions Thrown:

ArgumentException:
Condition: The DisplayCenter item of the window is specified as the active cursor.
Message: The DisplayCenter cannot be the active cursor.

3.5.2 ILockingWindow Methods

3.5.2.1 ILockingWindow.GetTimestamp

Description:

Gets the timestamp value of a specific item, such as a cursor or center of the data display.

Declaration Syntax:

Visual Basic

```
Function GetTimestamp (item As LockingWindowItem) As Decimal
```

C#

```
Decimal GetTimestamp (LockingWindowItem item)
```

C++

```
Decimal  GetTimestamp (LockingWindowItem item)
```

Arguments:

`item` – Selects which item to get the timestamp for.

Return Value:

A Decimal type value is returned that represents the timestamp in picoseconds.

Exceptions Thrown:

TlaNoDataException:
Condition: None of the data sources in the data window have any data for which timestamps can be defined.
Message: The window has no data displayed.

3.5.2.2 ILockingWindow.SetTimestamp

Description:

This method is used to set the timestamp value of the cursors and the display center of a data window.

Declaration Syntax:

Visual Basic

```
Sub SetTimestamp (screenItem As LockingWindowItem, value As Decimal)
```

C#

```
void SetTimestamp (LockingWindowItem screenItem, Decimal value)
```

C++

```
void SetTimestamp (LockingWindowItem screenItem, Decimal value)
```

Arguments:

```
screenItem – A member from the LockingWindowItem enumeration, which
       indicates which data window element is being set.
```

`value` – The timestamp value in picoseconds.

Return Value:

None.

Exceptions Thrown:

TlaNoDataException :
Condition: No data sources in the window have data.
Message: The window has no data displayed.

3.5.3 ILockingWindow Events

3.5.3.1 ILockingWindow.ItemChanged

Description:

Data windows that are capable of being locked raise this event when the user changes the timestamp value of one of the locked screen elements.

Declaration Syntax:

Visual Basic

```
Event ItemChanged As LockedItemChangedHandler
```

C#

```
event LockedItemChangedHandler ItemChanged;
```

C++

```
__event  LockedItemChangedHandler ItemChanged;
```

Event Data:

The event passes a single argument, a member of the LockingWindowItem enumeration, to the event handler. The argument specifies which data window item was changed, a cursor or the screen center for example.

3.6 ISearchableWindow

Description:

This interface is implemented by data windows that have named search definitions. In addition to the functionality of its base interface, ISearchableWindow allows clients to discover the names of all searches defined in a data window object; and this interfaces allows searches to be performed by name.

Namespace: Tektronix.LogicAnalyzer.Common

Declaration Syntax:

Visual Basic

```
Interface ISearchableWindow
      Inherits ILockingWindow
```

C#

```
interface ISearchableWindow : ILockingWindow
```

C++

```
__gc __interface ISearchableWindow : public ILockingWindow
```

3.6.1 ISearchableWindow Properties

3.6.1.1 ISearchableWindow.SearchNames

Description:

Gets an array of strings. Each string in the array is a search that is defined within the data window. The value of this propery is null when the data window has no searches defined.

Declaration Syntax:

Visual Basic

```
ReadOnly Property SearchNames As String()
```

C#

```
string[] SearchNames { get; }
```

C++

```
String* get_SearchNames () _gc[];
```

Arguments:

None.

Exceptions Thrown:

None

Remarks:

Individual elements in the array can be used as arguments to ISearchableWindow.PerformSearch.

Changing the elements in the string array has no effect on the names of searches defined in the data window.

3.6.2 ISearchableWindow Methods

3.6.2.1 ISearchableWindow.StartSearch

Description:

Calling this method causes a specified search to be performed in a specified direction. The search is specified by name, and the search is performed forward or backward from the active cursor. Search results are supplied by SearchCompleted event.

Declaration Syntax:

Visual Basic

```
Sub StartSearch (name As String, isForward As Boolean)
```

C#

```
void StartSearch (string name, bool isForward)
```

C++

```
void StartSearch (String* name, bool isForward)
```

Arguments:

`name` – The name of the search to be started. For listing and waveform windows, this is the name of the search as it appears in the Search Definitions dialog.

`isForward` – If this argument is true, then the search is performed forward from active cursor. If false, then a backward search is performed.

Return Value:

None.

Exceptions Thrown:

ArgumentNullException :
Condition: The name argument is null.
Message: <Determined by the object implementing the interface>.

ArgumentException :
Condition: There is no search definition that has the given name.
Message: <Determined by the object implementing the interface>.

Remarks:

Since searches don't necessarily complete immediately, this method initiates the given search and then returns. Clients should subscribe to the SearchCompleted event to be notified when the search is complete. Some searches can several second to several minutes to complete.

If a search is successful, the active cursor is moved to first sample in the data window that meets the search criteria. If cursor moves outside of the data display area, then the window scrolls until the active cursor is in the display center.

3.6.3 ISearchableWindow Events

3.6.3.1 ISearchableWindow.SearchCompleted

Description:

This event is raised when a searchable data window completes a search. The event data describes whether the search was sucessfu

Declaration Syntax:

Visual Basic

```
Event SearchCompleted As SearchCompletedHandler
```

C#

```
event SearchCompletedHandler SearchCompleted;
```

C++

```
__event SearchCompletedHandler SearchCompleted ;
```

Event Data:

Description of how this event uses its arguments if applicable.

Remarks:

Specific details about the event.

3.7 IDataSource

Description:

This is the base interface for all data sources, including Logic analyzer, DSO, and plug-in interfaces. IDataSource contains only those members that must be implemented by all data sources that are accessible through TPI.

Namespace: Tektronix.LogicAnalyzer.Common

Declaration Syntax:

Visual Basic

```
Interface IDataSource
      Inherits IValidity
```

C#

```
interface IDataSource
      : IValidity
```

C++

```
__gc __interface IDataSource
      : public IValidity
```

Remarks:

The purpose of this interface is to allow methods that deal with all kinds of data sources to reference them all with the same type, and it also to allows differentiation of data sources from other kinds of objects that might be held inside a collection.

3.7.1 IDataSource Properties

3.7.1.1 IDataSource.UserName

Description:

The user name of the data source as it should appear to users. For data sources that appear as icons in the system window, this is the name that appears with the icon.

Declaration Syntax:

Visual Basic

```
Property UserName As String
```

C#

```
string UserName { set; get;}
```

C++

```
String* get_UserName ();
void set_UserName (String*);
```

Arguments:

None.

Exceptions Thrown:

ArgumentNullException:
Condition: The property is set to a null reference.
Message: <The message string is implementation specific.>

ArgumentException:
Condition: The property is set name that is already used by another data source in the system.
Message: The given name is already in use by another data source.

Remarks:

Some implementations might allow the user name to be set to null without throwing an exception. Clients of this interface should assume that null user names are not allowed.

3.7.1.2 IDataSource.FileName

Description:

The file name of the data source as it exists.

Declaration Syntax:

Visual Basic

```
ReadOnly Property FileName As String
```

C#

```
string FileName { get;}
```

C++

```
String* get_FileName ();
```

Arguments:

None.

Exceptions Thrown:

None.

3.7.1.3 IDataSource.FilePath

Description:

The file path of the data source as it exists.

Declaration Syntax:

Visual Basic

```
ReadOnly Property FilePath As String
```

C#

```
string FilePath { get;}
```

C++

```
String* get_FilePath ();
```

Arguments:

None.

Exceptions Thrown:

None.

3.7.1.4 IDataSource.IsReference

Description:

Gets a Boolean indication whether the data source represents reference data from a saved file.

Declaration Syntax:

Visual Basic

```
ReadOnly Property IsReference As Boolean
```

C#

```
bool IsReference { get; }
```

C++

```
bool get_IsReference ();
```

Arguments:

None

Exceptions Thrown:

None

Remarks:

The data in reference data sources do not change. The data remains the same even after an acquisition.

The setup of a reference data source is usually read-only. The IsReadOnlySetup property can accessed to determine whether it is read-only.

3.7.1.5 IDataSource.IsReadOnlySetup

Description:

When the value of this property is true, the setup of the data source cannot be changed. Such a data source might still be able to provide data, but any attempt to change its setup will throw a InvalidOperationException.

Declaration Syntax:

Visual Basic

```
ReadOnly Property IsReadOnlySetup As Boolean
```

C#

```
bool IsReadOnlySetup { get; }
```

C++

```
bool get_IsReadOnlySetup ();
```

Arguments:

None

Exceptions Thrown:

None.

3.7.1.6 IDataSource.IsDataAvailable

Description:

Gets a Boolean value that indicates whether the data source can currently supply coherent data. When this value if false, any attempt to get data from the data source will fail.

Declaration Syntax:

Visual Basic

```
ReadOnly Property IsDataAvailable As Boolean
```

C#

```
bool IsDataAvailable { get; }
```

C++

```
bool get_IsDataAvailable ();
```

Arguments:

None.

Exceptions Thrown:

None.

3.7.2 IDataSource Events

3.7.2.1 IDataSource.UserNameChanged

Description:

Data sources fire this event when the UserName property of the object changes.

Declaration Syntax:

Visual Basic

```
Event UserNameChanged As EventHandler
```

C#

```
event EventHandler UserNameChanged;
```

C++

```
__event EventHandler UserNameChanged;
```

Event Data:

The UserName property of the data source will contain the new name.

Remarks:

The TLA requires that all data sources have unique names. It will subscribe to this event for every data source plug-in in the system. If a name change causes a duplicate name in the system, the TLA will set it to a unique name.

3.8 IStandardDataSource

Description:

This interface inherits from IDataSource and specifies a standard interface for data source objects to provide access to their data. TLA application implements this interface for LA, DSO and external oscilloscope objects.

Namespace: Tektronix.LogicAnalyzer.Common

Declaration Syntax:

Visual Basic

```
Interface IStandardDataSource
      Inherits IDataSource
```

C#

```
interface IStandardDataSource
      : IDataSource
```

C++

```
__gc __interface IStandardDataSource
      : public IDataSource
```

Remarks:

If a data source plug-in implements IStandardData source, clients of the plug-in will be able to access its data in a manner that is consistent with the representation of built-in modules such as ILAModule.

3.8.1 IStandardDataSource Methods

3.8.1.1 IStandardDataSource.GetChannelGroupingObject

Description:

This method is used to obtain an IChannelGrouping object. Clients can use the returned object to retrieve channel grouping information from the associated data source.

Declaration Syntax:

Visual Basic

```
Function GetChannelGroupingObject () As IChannelGrouping
```

C#

```
IChannelGrouping GetChannelGroupingObject ()
```

C++

```
IChannelGrouping* GetChannelGroupingObject ()
```

Arguments:

None.

Return Value:

A reference to an object of type IChannelGrouping is returned.

3.8.1.2 IStandardDataSource.GetAcquisitionDataObject

Description:

This method is used to obtain an IAcquisitionData object. Clients can use the returned object to retrieve information about the acquisition data acquired from the associated data source.

Declaration Syntax:

Visual Basic

```
Function GetAcquisitionDataObject () As IAcquisitionData
```

C#

```
IAcquisitionData GetAcquisitionDataObject ()
```

C++

```
IAcquisitionData* GetAcquisitionDataObject ()
```

Arguments:

None.

Return Value:

A reference to an object of type IAcquisitionData is returned.

Exceptions Thrown:

Exception-Class-Name :
> *Condition*: Condition under which the exception is thrown.
> *Message*: Message string associated with exception.

3.9 IChannelGrouping

Description:

This interface provides common access to the channel grouping setup of both LA and DSO instruments. It allows access to the setup on the basis of groups and bits. These interfaces provide query mechanisms to see if particular objects or operations are allowed for the specific instrument. These queries can be used to allow for different capabilities between LA and DSO instruments.

Namespace: Tektronix.LogicAnalyzer.TpiNet

Declaration Syntax:

Visual Basic

```
Interface IChannelGrouping
```

C#

```
interface IChannelGrouping
```

C++

```
__gc __interface IChannelGrouping
```

Remarks:

Naming conventions for objects and methods are admittedly a little awkward for the DSO. “Groups” are meant to refer to multi-bit entities. While “Groups” seems natural enough to LA clients, DSO clients might be expecting “Channels”. In order to keep the LA and DSO interfaces as similar as possible, I’ve avoided the “channel” terminology because it means a single bit for the LA and 16 bits for the DSO. The following terminology guide may help clarify matters:

“Group” – a multi-bit entity: a traditional LA group or a traditional DSO channel.
“Bit” – a single bit entity: a traditional LA channel or a single bit of a DSO channel.

3.9.1 IChannelGrouping Properties

3.9.1.1IChannelGrouping.Groups

Description:

Gets a list of all channel groups in the instrument. For the LA, the list contains channel groups. For the DSO, the list contains 16-bit channels.

Declaration Syntax:

Visual Basic

```
ReadOnly Property Groups As ArrayList
```

C#

```
ArrayList Groups { get;}
```

C++

```
ArrayList* get_Groups ();
```

Arguments:

None.

Exceptions Thrown:

InvalidOperationException :
Condition: Attempt is made to change the list.
Message: Group list is read-only.

Remarks:

The value of this property is a read-only ArrayList object. Although getting the property value never throws an exception, any attempt to modify the list will throw an exception. This means that the number of available groups in the instrument cannot be changed by operations on this ArrayList. Groups can be added or removed by the following IChannelGrouping methods: AddGroup and RemoveGroup.

3.9.1.2 IChannelGrouping.Bits

Description:

Gets a list of all single bit data entities in the instrument. For the LA, the list contains all acquisition channels. For the DSO, the list contains all of the individual bits for all of the 16-bit channels.

Declaration Syntax:

Visual Basic

```
ReadOnly Property Bits As ArrayList
```

C#

```
ArrayList Bits { get;}
```

C++

```
ArrayList* get_Bits ();
```

Arguments:

None.

Exceptions Thrown:

InvalidOperationException :
    Condition: Attempt is made to change the list.
    Message: Channel list is read-only.

Remarks:

The value of this property is a read-only ArrayList object. Although getting the property value never throws an exception, any attempt to modify the list will throw an exception. This means that the number of available bits in the instrument cannot be changed by operations on this ArrayList.

3.9.2 IChannelGrouping Methods

3.9.2.1 IChannelGrouping.CreateGroup

Description:

This method is used to create a new channel group for the instrument. The group is not actually added to the instrument. It must be initialized by appropriate IChannelGroup methods and then added to the instrument with IChannelGrouping.AddGroup().

Declaration Syntax:

Visual Basic

```
Function CreateGroup (userName As String) As IChannelGroup
```

C#

```
IChannelGroup CreateGroup (string userName)
```

C++

```
IChannelGroup* CreateGroup (String* userName)
```

Arguments:

userName – the name of the new group.

Return Value:

Returns an IChannelGroup reference with the given name.

Exceptions Thrown:

InvalidOperationException :
  Condition: Attempt is made to create group on a read only setup.
  Message: Instrument setup is read-only.

InvalidOperationException :
  Condition: Attempt is made to create group on a DSO.
  Message: Instrument channel grouping is read-only.

Remarks:

Clients should check the IDataSource.IsReadOnlySetup property before adding or removing LA groups.

3.9.2.2 IChannelGrouping.AddGroup

Description:

This method is used to add a pre-initialized channel group to the instrument. The group should be created with IChannelGrouping.CreateGroup() and initialized by appropriate IChannelGroup methods before being added to the instrument with IChannelGrouping.AddGroup().

Declaration Syntax:

Visual Basic

```
Sub AddGroup (group As IChannelGroup)
```

C#

```
void AddGroup (IChannelGroup group)
```

C++

```
void AddGroup (IChannelGroup* group)
```

Arguments:

group – the pre-initialized group to be added.

Return Value:

None.

Exceptions Thrown:

InvalidOperationException :
> *Condition*: Attempt is made to create group on a read only setup.
> *Message*: Instrument setup is read-only.

InvalidOperationException :
> *Condition*: Attempt is made to create group on a DSO.
> *Message*: Instrument channel grouping is read-only.

Remarks:

Addition of a group to the channel grouping fires the GroupsChanged event.

Clients should check the IDataSource.IsReadOnlySetup property before adding or removing LA groups.

3.9.2.3 IChannelGrouping.RemoveGroup

Description:

This method is used to remove a channel group from the instrument.

Declaration Syntax:

Visual Basic

```
Sub RemoveGroup (userName As String)
```

C#

```
void RemoveGroup (string userName)
```

C++

```
void RemoveGroup (String* userName)
```

Arguments:

userName – the name of the group to be removed.

Return Value:

None.

Exceptions Thrown:

InvalidOperationException :
Condition: Attempt is made to remove group on a read only setup.
Message: Instrument setup is read-only.

InvalidOperationException :
Condition: Attempt is made to remove group on a DSO.
Message: Instrument channel grouping is read-only.

ArgumentException :
Condition: No group with specified name exists.
Message: Specified group does not exist.

Remarks:

Removal of a group from the channel grouping fires the GroupsChanged event.

Clients should check the IDataSource.IsReadOnlySetup property before adding or removing LA groups.

3.9.3 IChannelGrouping Events

3.9.3.1 IChannelGrouping.GroupsChanged

Description:

Instruments that support addition and removal of channel groups raise this event when a group is added or removed from the channel grouping setup.

Declaration Syntax:

Visual Basic

```
Event GroupsChanged As ObjectEventHandler
```

C#

```
event ObjectEventHandler GroupsChanged;
```

C++

```
__event ObjectEventHandler GroupsChanged;
```

Event Data:

The IChannelGrouping object is passed as a parameter.

3.10 IChannelGroup

Description:

This interface provides common access to a channel group for both LA and DSO instruments.

Namespace: Tektronix.LogicAnalyzer.TpiNet

Declaration Syntax:

Visual Basic

```
Interface IChannelGroup
```

C#

```
interface IChannelGroup
```

C++

```
__gc __interface IChannelGroup
```

3.10.1 IChannelGroup Properties

3.10.1.1 IChannelGroup.Name

Description:

The Name property provides the group with a modifiable name.

Declaration Syntax:

Visual Basic

```
Property Name As String
```

C#

```
string Name { get; set;}
```

C++

```
String* get_Name ();
void set_Name ( String* name );
```

Arguments:

name – a new name for the group supplied to the setter.

Exceptions Thrown:

InvalidOperationException :
    Condition: Attempt is made to rename group on a read only setup.
    Message: Instrument setup is read-only.
InvalidOperationException :
    Condition: Attempt is made to rename group that already exists.
    Message: Name already exists.

Remarks:

Changing the group name raises the NameChanged event of the group.

Clients should check the IDataSource.IsReadOnlySetup property before renaming a group.

3.10.1.2 IChannelGroup.NumberOfBits

Description:

The read-only NumberOfBits property provides the group data width in bits.

Declaration Syntax:

Visual Basic

```
ReadOnly Property NumberOfBits As Short
```

C#

```
short NumberOfBits { get; }
```

C++

```
short get_NumberOfBits ();
```

Arguments:

None.

Exceptions Thrown:

None.

Remarks:

None.

3.10.1.3 IChannelGroup.BitsAssigned

Description:

The BitsAssigned property represents the bits belonging to the group as a comma separated list of bit names or bit ranges.

Declaration Syntax:

Visual Basic

```
Property BitsAssigned As String
```

C#

```
string BitsAssigned { get; set;}
```

C++

```
String* get_BitsAssigned ();
Void set_BitsAssigned( String* bits );
```

Arguments:

Bits – the bits belonging to the group in the form of a comma separated list of bit names or bit ranges. For the DSO, bit names will have the form “ChN(15-0)”, where “N” is 1 through number of DSO channels. For the LA, bit names may be channel hardware names (“CK0” or “A3(0)”), user defined channel names (“RESET”), section names “A3”, or section sub ranges (“A3(7-4)”). Most significant bits are listed first and the least significant bit is listed last.

Exceptions Thrown:

InvalidOperationException :
Condition: Attempt is made to set bits of group on a read only setup.
Message: Instrument setup is read-only.

InvalidOperationException :
Condition: Attempt is made to set bits of group on a DSO.
Message: Instrument channel grouping is read-only.

ArgumentException :
Condition: An unrecognized channel was specified.
Message: Specified channel does not exist.

Remarks:

Clients should check the IDataSource.IsReadOnlySetup property before assigning channels to a LA group.

3.10.1.4 IChannelGroup.TimeOffset

Description:

The TimeOffset property allows DSO channel groups to have their time alignment skewed for time correlation purposes.

Declaration Syntax:

Visual Basic

```
Property TimeOffset () As Decimal
```

C#

```
Decimal TimeOffset () { get; set; }
```

C++

```
Decimal get_TimeOffset ();
void set_TimeOffset(Decimal offset );
```

Arguments:

offset – The amount of time in picoseconds to skew the DSO channel relative to others in the instrument.

Exceptions Thrown:

InvalidOperationException :
Condition: Attempt is made to set time alignment offset of group on a read only setup.
Message: Instrument setup is read-only.

InvalidOperationException :
Condition: Attempt is made to set time alignment offset of group on a LA.
Message: Instrument setup operation is not supported.

Remarks:

Clients should check the IDataSource.IsReadOnlySetup property before modifying the time alignment offset of a DSO channel group.

3.10.2 IChannelGroup Methods

3.10.2.1 IChannelGroup.BitOfGroup

Description:

The BitOfGroup method provides access to the individual channel bits comprising the group. The IChannelBit object returned may not be reassigned, but its methods and properties may be used to modify the setup (for example, changing the channel polarity).

Declaration Syntax:

Visual Basic

```
Function BitOfGroup ( bitNumber As Short ) As IChannelBit
```

C#

```
IChannelBit BitOfGroup ( short bitNumber )
```

C++

```
IChannelBit* BitOfGroup ( short bitNumber )
```

Arguments:

bitNumber – The bitNumber specifies which bit of the group to retrieve. Bit numbers start at zero for the LSB.

Return value:

A reference to the IChannelBit object specified by bitNumber is returned

Exceptions Thrown:

ArgumentOutOfRangeException :
  Condition: An unrecognized bit number was specified.
  Message: Specified group bit does not exist.

3.10.3 IChannelGroup Events

3.10.3.1 IChannelGroup.NameChanged

Description:

When the name of a channel group is changed, the NameChanged event is raised.

Declaration Syntax:

Visual Basic

```
Event NameChanged As ObjectEventHandler
```

C#

```
event ObjectEventHandler NameChanged;
```

C++

```
__event ObjectEventHandler NameChanged;
```

Event Data:

The IChannelGroup object is passed as a parameter.

3.10.3.2 IChannelGroup.TimeOffsetChanged

Description:

When the Time Offset value of the channel group is changed, the NameChanged event is raised.

Declaration Syntax:

Visual Basic

```
Event TimeOffsetChanged As ObjectEventHandler
```

C#

```
event ObjectEventHandler TimeOffsetChanged;
```

C++

```
__event ObjectEventHandler TimeOffsetChanged;
```

Event Data:

The IChannelGroup object is passed as a parameter.

3.10.3.3 IChannelGroup.BitsAssignedChanged

Description:

When the channel composition of a group is changed, the BitsAssignedChanged event is raised. Changing the user names of bits belonging to the group raises this event as well (the channel composition didn’t change, but the string representation did).

Declaration Syntax:

Visual Basic

```
Event BitsAssignedChanged As ObjectEventHandler
```

C#

```
event ObjectEventHandler BitsAssignedChanged;
```

C++

```
__event ObjectEventHandler BitsAssignedChanged;
```

Event Data:

The IChannelGroup object is passed as a parameter.

3.10.3.4 IChannelGroup.PolarityChanged

Description:

When the polarity of a bit belonging to the group is changed, the PolarityChanged event is raised. Even though the polarity is not a direct property of the group, polarity change events for the bits belonging to the group are funneled through the group as a convenience to clients.

Declaration Syntax:

Visual Basic

```
Event PolarityChanged As ObjectEventHandler
```

C#

```
event ObjectEventHandler PolarityChanged;
```

C++

```
__event ObjectEventHandler PolarityChanged;
```

Event Data:

The IChannelGroup object is passed as a parameter (not the IChannelBit which actually changed).

3.10.3.5 IChannelGroup.CompareEnableChanged

Description:

When the compare enable of a bit belonging to the group is changed, the CompareEnableChanged event is raised. Even though the compare enable is not a direct property of the group, compare enable events for the bits belonging to the group are funneled through the group as a convenience to clients.

Declaration Syntax:

Visual Basic

```
Event CompareEnableChanged As ObjectEventHandler
```

C#

```
event ObjectEventHandler CompareEnableChanged;
```

C++

```
__event ObjectEventHandler CompareEnableChanged;
```

Event Data:

The IChannelGroup object is passed as a parameter (not the IChannelBit which actually changed).

3.11 IChannelBit

Description:

This interface provides common access to a channel bit for both LA and DSO instruments.

Namespace: Tektronix.LogicAnalyzer.TpiNet

Declaration Syntax:

Visual Basic

```
Interface IChannelBit
```

C#

```
interface IChannelBit
```

C++

```
__gc __interface IChannelBit
```

3.11.1 IChannelBit Properties

3.11.1.1 IChannelBit.Name

Description:

The Name property provides the bit with a modifiable name.

Declaration Syntax:

Visual Basic

```
Property Name As String
```

C#

```
string Name { get; set;}
```

C++

```
String* get_Name ();
void set_Name ( String* name );
```

Arguments:

name – a new name for the bit supplied to the setter.

Exceptions Thrown:

InvalidOperationException :
    Condition: Attempt is made to rename a bit on a read only setup.
    Message: Instrument setup is read-only.

InvalidOperationException:
    Condition: Attempt is made to rename a bit on a DSO.
    Message: Instrument setup operation is not supported.

Remarks:

Changing the bit name raises the NameChanged event of the bit.

The IDataSource.IsReadOnlySetup property should be checked before attempting to modify the name of a LA bit.

The DSO returns the hardware bit name as the name of the bit always.

3.11.1.2 IChannelBit.HardwareName

Description:

The read-only HardwareName property provides the hardware name of the bit.

Declaration Syntax:

Visual Basic

```
ReadOnly Property HardwareName As String
```

C#

```
string HardwareName { get; }
```

C++

```
String* get_HardwareName ();
```

Arguments:

None.

Exceptions Thrown:

None.

Remarks:

The DSO returns hardware names in the form ChN(M) where N is channel number containing the bit and M is the bit number (0 is LSB).

3.11.1.3 IChannelBit.Polarity

Description:

The Polarity property provides the bit with a modifiable polarity.

Declaration Syntax:

Visual Basic

```
Property Polarity As PolarityValue
```

C#

```
PolarityValue Polarity { get; set;}
```

C++

```
PolarityValue get_Polarity ();
void set_Polarity ( PolarityValue newPolarity );
```

Arguments:

newPolarity – desired polarity for the bit.

Exceptions Thrown:

InvalidOperationException:
Condition: Attempt is made to set polarity of a bit on a read only setup.
Message: Instrument setup is read-only.

InvalidOperationException:
Condition: Attempt is made to set polarity of a bit on a DSO.
Message: Instrument setup operation is not supported.

Remarks:

Changing the bit polarity raises the PolarityChanged event of the bit.

The IDataSource.IsReadOnlySetup property should be checked before attempting to modify the polarity of a LA bit.

The DSO polarity getter always returns Positive.

3.11.1.4 IChannelBit.CompareEnable

Description:

The CompareEnable property provides the bit with a modifiable compare enable flag.

Declaration Syntax:

Visual Basic

```
Property CompareEnable As Boolean
```

C#

```
bool CompareEnable { get; set;}
```

C++

```
bool get_CompareEnable ();
void set_CompareEnable (bool enable);
```

Arguments:

enable – desired compare enable for the bit.

Exceptions Thrown:

InvalidOperationException:
Condition: Attempt is made to set compare enable of a bit on a read only setup.
Message: Instrument setup is read-only.

InvalidOperationException:
Condition: Attempt is made to set compare enable of a bit on a DSO.
Message: Instrument setup operation is not supported.

Remarks:

Changing the bit compare enable raises the CompareEnableChanged event of the bit.

The IDataSource.IsReadOnlySetup property should be checked before attempting to modify the compare enable of a LA bit.

The DSO compare enable getter always returns false.

3.11.1.5 IChannelBit.CanGroup

Description:

The CanGroup property indicates whether this Channel Bit can be included within a group definition.

Declaration Syntax:

Visual Basic

```
Property CanGroup As Boolean
```

C#

```
bool CanGroup { get }
```

C++

```
bool get_CanGroup ();
```

Arguments:

None.

Exceptions Thrown:

None.

3.11.1.1 IChannelBit.FromProbe

Description:

The FromProbe property indicates whether this Channel Bit comes from a probe connector. Some internal channel bits (example: Status bits) do not come from probe connectors

Declaration Syntax:

Visual Basic

```
Property FromProbe As Boolean
```

C#

```
bool FromProbe { get }
```

C++

```
bool get_ FromProbe ();
```

Arguments:

None.

Exceptions Thrown:

None.

3.11.2 IChannelBit Events

3.11.2.1 IChannelBit.NameChanged

Description:

When the name of a channel bit is changed, the NameChanged event is raised.

Declaration Syntax:

Visual Basic

```
Event NameChanged As ObjectEventHandler
```

C#

```
event ObjectEventHandler NameChanged;
```

C++

```
__event ObjectEventHandler NameChanged;
```

Event Data:

The IChannelBit object is passed as a parameter.

3.11.2.2 IChannelBit.PolarityChanged

Description:

When the polarity of a channel bit is changed, the PolarityChanged event is raised.

Declaration Syntax:

Visual Basic

```
Event PolarityChanged As ObjectEventHandler
```

C#

```
event ObjectEventHandler PolarityChanged;
```

C++

```
__event ObjectEventHandler PolarityChanged;
```

Event Data:

The IChannelBit object is passed as a parameter.

3.11.2.3 IChannelBit.CompareEnableChanged

Description:

When the compare enable of a bit is changed, the CompareEnableChanged event is raised.

Declaration Syntax:

Visual Basic

```
Event CompareEnableChanged As ObjectEventHandler
```

C#

```
event ObjectEventHandler CompareEnableChanged;
```

C++

```
__event ObjectEventHandler CompareEnableChanged;
```

Event Data:

The IChannelBit object is passed as a parameter.

3.12 IAcquisitionData

Description:

This interface provides access to read only properties and methods that reflect various setup parameters that were in effect at the time acquisition data was acquired. It provides events and flags to indicate when acquisition data is valid and provides a means to create an IRecordAccess reader object for getting acquisition records. This interface is common to LA, DSO, and iView modules.

Namespace: Tektronix.LogicAnalyzer.TpiNet

Declaration Syntax:

Visual Basic

```
Interface IAcquisitionData Inherits IValidity
```

C#

```
interface IAcquisitionData : IValidity
```

C++

```
__gc __interface IAcquisitionData : public IValidity
```

Remarks:

The IValidity.ValidityChanged event is used to signal when the acquisition data has changed. The IValidity.IsValid property may be checked to see if new acquisition data is available (true) or if prior acquisition data has become unavailable (false).

3.12.1 IAcquisitionData Properties

3.12.1.1 IAcquisitionData.AcquisitionDate

Description:

The read-only AcquisitionDate property indicates when the acquisition data was acquired. This property is a System.DateTime structure as defined by the framework and includes both date and time information.

Declaration Syntax:

Visual Basic

```
ReadOnly Property AcquisitionDate As DateTime
```

C#

```
DateTime AcquisitionDate { get; }
```

C++

```
DateTime get_AcquisitionDate ();
```

Arguments:

None.

Exceptions Thrown:

InvalidOperationException :
Condition: Status requested when status IsValid property is false.
Message: Module status values are not valid at this time.

Remarks:

If the module has not been used to acquire data, the current time of day is returned.

3.12.1.2 IAcquisitionData.TriggerReason

Description:

The read-only TriggerReason property indicates how the main timebase of the module was triggered.

Declaration Syntax:

Visual Basic

```
ReadOnly Property TriggerReason As TriggerReasonValue
```

C#

```
TriggerReasonValue TriggerReason { get; }
```

C++

```
TriggerReasonValue get_TriggerReason ();
```

Arguments:

None.

Exceptions Thrown:

InvalidOperationException :
Condition: Status requested when status IsValid property is false.
Message: Module status values are not valid at this time.

3.12.1.3 IAcquisitionData.SystemTriggerTime

Description:

The read-only SystemTriggerTime property provides the system trigger timestamp in picoseconds.

Declaration Syntax:

Visual Basic

```
ReadOnly Property SystemTriggerTime As Decimal
```

C#

```
Decimal SystemTriggerTime { get; }
```

C++

```
Decimal get_SystemTriggerTime ();
```

Arguments:

None.

Exceptions Thrown:

InvalidOperationException :
Condition: Status requested when status IsValid property is false.
Message: Module status values are not valid at this time.

3.12.1.4 IAcquisitionData.FrameTimeOffset

Description:

The FrameTimeOffset property holds a propagation time value in picoseconds for the expansion mainframe in which the instrument resides. This value is zero for all acquisition modules within the TLA600 series mainframe. This value is also zero for acquisition modules residing in the TLA700 series controller mainframe. This value is non-zero for acquisition modules residing the expansion frames of the TLA700 series.

Declaration Syntax:

Visual Basic

```
ReadOnly Property FrameTimeOffset As Decimal
```

C#

```
Decimal FrameTimeOffset { get; }
```

C++

```
Decimal get_FrameTimeOffset ();
```

Arguments:

None.

Exceptions Thrown:

InvalidOperationException :
Condition: Status requested when status IsValid property is false.
Message: Module status values are not valid at this time.

3.12.1.5 IAcquisitionData.TimeOffset

Description:

The TimeOffset property allows a user specified time adjustment to be made to module data for purposes of correlation. The TimeOffset is a picosecond amount of time to shift each acquisition sample relative to its raw unbiased acquisition time.

Declaration Syntax:

Visual Basic

```
Property TimeOffset As Decimal
```

C#

```
Decimal TimeOffset { get; set; }
```

C++

```
Decimal get_TimeOffset ();
void set_TimeOffset ( Decimal offset );
```

Arguments:

None.

Exceptions Thrown:

InvalidOperationException :
Condition: Status requested when status IsValid property is false.
Message: Module status values are not valid at this time.

InvalidOperationException :
Condition: Attempt to modify TimeOffset when IDataSource.IsReadOnlySetup is true.
Message: Module setup is read only.

3.12.1.6 IAcquisitionData.CorrelationTimeOffset

Description:

The read-only CorrelationTimeOffset property provides the ICorrelator with a time adjustment value in picoseconds for the data source. This value is equal to TimeOffset – SystemTriggerTime. The CorrelationTimeOffset should be added to timestamps reported by modules before the module timestamp is passed into the ICorrelator. The CorrelationTimeOffset should also be added to a module timestamp before being passed to ILockingWindow.SetTimestamp.

Declaration Syntax:

Visual Basic

```
ReadOnly Property CorrelationTimeOffset As Decimal
```

C#

```
Decimal CorrelationTimeOffset { get; }
```

C++

```
Decimal get_CorrelationTimeOffset ();
```

Arguments:

None.

Exceptions Thrown:

InvalidOperationException :
Condition: Status requested when status IsValid property is false.
Message: Module status values are not valid at this time.

3.12.2 IAcquisitionData Methods

3.12.2.1 IAcquisitionData.CreateRecordAccessObject

Description:

This method is used to obtain an IRecordAccess object. Clients can use the returned object to retrieve data samples from the associated data source.

Declaration Syntax:

Visual Basic

```
Function CreateRecordAccessObject () As IRecordAccess
```

C#

```
IRecordAccess CreateRecordAccessObject ()
```

C++

```
IRecordAccess* CreateRecordAccessObject ()
```

Arguments:

None.

Return Value:

A reference to an object of type IRecordAcces is returned.

Exceptions Thrown:

InvalidOperationException :
Condition: Called when IsValid is false.
Message: Module status values are not valid at this time.

Remarks:

Clients should call Dispose() on the IRecordAccess object when finished. See IRecordAccess for details.

3.12.2.2 IAcquisitionData.IsDataSetSupported

Description:

This method is used to see if the instrument supports the designated data set.

Declaration Syntax:

Visual Basic

```
Function IsDataSetSupported ( DataSetValue which ) As Boolean
```

C#

```
bool IsDataSetSupported ( DataSetValue which )
```

C++

```
bool IsDataSetSupported ( DataSetValue which )
```

Arguments:

which – a data set to check.

Return Value:

A return of true means the instrument supports the designated data set. A return of false means the instrument does not support the designated data set.

Exceptions Thrown:

None

Remarks:

Even though an instrument may return true to indicate support of a data set, it does not follow that acquisition data exists for that data set.

Examples:

```
C# Example:
public Int64 MagniVuSamples( IAcquisitionData acqData )
{
      DataSetValue dset = DataSetValue.MagniVu;
      if (! acqData.isValid) return 0;
      if (! acqData.IsDataSetSupported(dset)) return 0;
      return acqData.TimebaseSummary(dset).NumberOfSamples;
}
```

3.12.2.3 IAcquisitionData.TimebaseSummary

Description:

This method returns an object that summarizes the data captured for a particular timebase. An overloaded version takes a timebase value as a parameter to return a specific timebase. Another overloaded version that takes no parameters returns the Main timebase. This method should be called for each acquisition to get new summary information for that acquisition.

Declaration Syntax:

Visual Basic

```
Function TimebaseSummary ( dset As DataSetValue ) As AcquisitionSummary
Function TimebaseSummary () As AcquisitionSummary
```

C#

```
AcquisitionSummary TimebaseSummary (DataSetValue dset)
AcquisitionSummary TimebaseSummary ()
```

C++

```
AcquisitionSummary * TimebaseSummary (DataSetValue dset)
AcquisitionSummary * TimebaseSummary ()
```

Arguments:

dset – the timebase of interest.

Return Value:

Returns an `AcquisitionSummary` object to describe what was acquired for the timebase.

Exceptions Thrown:

InvalidOperationException :
    Condition: Status requested when status IsValid property is false.
    Message: Module status values are not valid at this time.

ArgumentException :
    Condition: Status for the given time base is not supported.
    Message: Status for the given time base is not supported.

Remarks:

The IAcquisitionData.IsValid property should be checked before requesting a timebase summary.

Examples:

```
C# Example:
public Int64 MagniVuSamples( IAcquisitionData acqData )
{
      DataSetValue dset = DataSetValue.MagniVu;
      if (! acqData.isValid) return 0;
      if (! acqData.IsDataSetSupported(dset)) return 0;
      return acqData.TimebaseSummary(dset).NumberOfSamples;
}
```

3.12.2.4 IAcquisitionData.GetTimestamp

Description:

This method returns the timestamp of a sample in picoseconds. The returned value does not include any time offsets (see TimeOffset and CorrelationTimeOffset).

Declaration Syntax:

Visual Basic

```
Function GetTimestamp (Long sample, dset As DataSetValue) As Decimal
```

C#

```
Decimal GetTimestamp (long sample, DataSetValue dset)
```

C++

```
Decimal GetTimestamp (__int64 sample, DataSetValue dset)
```

Arguments:

sample – the sample number of interest.
dset – the timebase of interest.

Return Value:

Returns the timestamp for the sample in picoseconds.

Exceptions Thrown:

InvalidOperationException :
Condition: Status requested when status IsValid property is false.
Message: Module status values are not valid at this time.

ArgumentException :
Condition: Status for the given time base is not supported.
Message: Status for the given time base is not supported.

ArgumentOutOfRangeException :
Condition: The data set has no such sample number.
Message: Invalid sample number.

Remarks:

The IAcquisitionData.IsValid property should be checked before requesting a timestamp.

Examples:

```
C# Example:
public class TimestampCache
{
      private IAcquisitionData acqData;
```

```
	private long sampleMRU;
	private DataSetValue dataSetMRU;
	private decimal timestampMRU;

	public TimestampCache( IAcquisitionData acquisitionData )
	{
		acqData = acquisitionData;
		sampleMRU = -1;
		dataSetMRU = DataSetValue.Main;
		timestampMRU = -1;
	}

	public UpdateCache( long sample, DataSetValue dset )
	{
		try {
			timestampMRU = acqData.GetTimestamp( sample, dset );
			sampleMRU = sample;
			dataSetMRU = dset;
		}
		catch (…) {}
	}
}
```

3.12.2.5 IAcquisitionData.GetSampleOnOrAfter

Description:

This method returns the sample number for the data set that occurs at or after the specified time. All time values are unbiased (they do not include TimeOffset or CorrelationTimeOffset).

Declaration Syntax:

Visual Basic

```
Function GetSampleOnOrAfter (time As Decimal, dset As DataSetValue,
[Out]nearest As Decimal) As Long
```

C#

```
long GetSampleOnOrAfter (Decimal time, DataSetValue dset, out Decimal
nearest)
```

C++

```
__int64 GetSampleOnOrAfter (Decimal time, DataSetValue dset, Decimal*
nearest)
```

Arguments:

time – the time of interest.
dset – the timebase of interest.
nearest – the timestamp of sample found is returned through this parameter.

Return Value:

Returns the sample number that occurs at or after the given time and places the timestamp of that sample in nearest. When no such sample exists, this returns –1 and sets nearest to zero.

Exceptions Thrown:

InvalidOperationException :
Condition: Status requested when status IsValid property is false.
Message: Module status values are not valid at this time.

ArgumentNullException:
Condition: Reference to the nearest parameter is null.
Message: Name of the null parameter: nearest.

ArgumentException :
Condition: Status for the given time base is not supported.
Message: Status for the given time base is not supported.

Remarks:

The IAcquisitionData.IsValid property should be checked before using this method.

Examples:

```
C# Example:
public class ForwardTimestampCache
{
      private IAcquisitionData acqData;
      private long sampleMRU;
      private decimal timestampMRU;

      public ForwardTimestampCache( IAcquisitionData acquisitionData )
      {
            acqData = acquisitionData;
            sampleMRU = -1;
            timestampMRU = -1;
      }

      public UpdateCacheAtOrAfter( decimal timestamp )
      {
            try {
                  sampleMRU = acqData.GetTimestampOnOrAfter(
                        timestamp, DataSetValue.Main, timestampMRU );
            }
            catch (…) {}
      }
}
```

3.12.2.6 IAcquisitionData.GetSampleOnOrBefore

Description:

This method returns the sample number for the data set that occurs at or before the specified time. All time values are unbiased (they do not include TimeOffset or CorrelationTimeOffset).

Declaration Syntax:

Visual Basic

```
Function GetSampleOnOrBefore (time As Decimal, dset As DataSetValue,
[Out]nearest As Decimal) As Long
```

C#

```
long GetSampleOnOrBefore (Decimal time, DataSetValue dset, out Decimal
nearest)
```

C++

```
__int64 GetSampleOnOrBefore (Decimal time, DataSetValue dset, Decimal*
nearest)
```

Arguments:

time – the time of interest.
dset – the timebase of interest.
nearest – the timestamp of sample found is returned through this parameter.

Return Value:

Returns the sample number that occurs at or before the given time and places the timestamp of that sample in nearest. When no such sample exists, this returns –1 and sets nearest to zero.

Exceptions Thrown:

InvalidOperationException :
Condition: Status requested when status IsValid property is false.
Message: Module status values are not valid at this time.

ArgumentNullException:
Condition: Reference to the nearest parameter is null.
Message: Name of the null parameter: nearest.

ArgumentException :
Condition: Status for the given time base is not supported.
Message: Status for the given time base is not supported.

Remarks:

The IAcquisitionData.IsValid property should be checked before using this method.

Examples:

3.12.3 IAcquisitionData Events

3.12.3.1 IAcquisitionData.TimeOffsetChanged

Description:

When the TimeOffset property is changed, the TimeOffsetChanged event is raised.

Declaration Syntax:

Visual Basic

```
Event TimeOffsetChanged As ObjectEventHandler
```

C#

```
event ObjectEventHandler TimeOffsetChanged;
```

C++

```
__event ObjectEventHandler TimeOffsetChanged;
```

Event Data:

The IAcquisitionData object is passed as a parameter.

3.13 AcquisitionSummary

Description:

This class provides timebase summary information for the last acquisition. This summary is obtained from the IAcquisitionData.TimebaseSummary() method. A new summary should be obtained when the instrument has new acquisition data. The IModuleRunStatus interface provides a RunStateChanged event and RunState property that can be used to determine when an updated summary is available.

Namespace: Tektronix.LogicAnalyzer.TpiNet

Declaration Syntax:

Visual Basic

```
[Serializable]
Class AcquisitionSummary
```

C#

```
[Serializable]
class AcquisitionSummary
```

C++

```
[Serializable]
__gc class AcquisitionSummary
```

3.13.1 AcquisitionSummary Properties

3.13.1.1 AcquisitionSummary.NumberOfSamples

Description:

The read-only NumberOfSamples property indicates the number of samples acquired.

Declaration Syntax:

Visual Basic

```
ReadOnly Property NumberOfSamples As Long
```

C#

```
long NumberOfSamples { get; }
```

C++

```
__int64 get_NumberOfSamples ();
```

Arguments:

None.

Exceptions Thrown:

None.

Remarks:

This property can be checked to see if any samples exist before accessing other properties that might generate exceptions when no data is available.

3.13.1.2 AcquisitionSummary.HasTriggerSample

Description:

The read-only HasTriggerSample property indicates if the timebase has a sample labeled as the trigger point.

Declaration Syntax:

Visual Basic

```
ReadOnly Property HasTriggerSample As Boolean
```

C#

```
bool HasTriggerSample { get; }
```

C++

```
bool get_HasTriggerSample ();
```

Arguments:

None.

Exceptions Thrown:

None.

Remarks:

This property can be checked to see if a trigger sample exists before getting the TriggerSample and TriggerTimestamp properties.

3.13.1.3 AcquisitionSummary.TriggerSample

Description:

The read-only TriggerSample property indicates which sample is labeled as the trigger point.

Declaration Syntax:

Visual Basic

```
ReadOnly Property TriggerSample As Long
```

C#

```
long TriggerSample { get; }
```

C++

```
__int64 get_TriggerSample ();
```

Arguments:

None.

Exceptions Thrown:

InvalidOperationException :
- *Condition*: No trigger sample exists.
- *Message*: No trigger sample exists.

Remarks:

The HasTriggerSample property should be checked before accessing this property.

3.13.1.4 AcquisitionSummary.BeginTimestamp

Description:

The read-only BeginTimestamp property returns the timestamp in picoseconds of the first sample acquired.

Declaration Syntax:

Visual Basic

```
ReadOnly Property BeginTimestamp As Decimal
```

C#

```
Decimal BeginTimestamp { get; }
```

C++

```
Decimal get_BeginTimestamp ();
```

Arguments:

None.

Exceptions Thrown:

InvalidOperationException :
Condition: No samples were acquired.
Message: No beginning timestamp exists.

Remarks:

The NumberOfSamples property should be checked before accessing this property.

3.13.1.5 AcquisitionSummary.EndTimestamp

Description:

The read-only EndTimestamp property returns the timestamp in picoseconds of the last sample acquired.

Declaration Syntax:

Visual Basic

```
ReadOnly Property EndTimestamp As Decimal
```

C#

```
Decimal EndTimestamp { get; }
```

C++

```
Decimal get_EndTimestamp ();
```

Arguments:

None.

Exceptions Thrown:

InvalidOperationException :
> *Condition*: No samples were acquired.
> *Message*: No ending timestamp exists.

Remarks:

The NumberOfSamples property should be checked before accessing this property.

3.13.1.6 AcquisitionSummary.TriggerTimestamp

Description:

The read-only TriggerTimestamp property returns the timestamp in picoseconds of the trigger point.

Declaration Syntax:

Visual Basic

```
ReadOnly Property TriggerTimestamp As Decimal
```

C#

```
Decimal TriggerTimestamp { get; }
```

C++

```
Decimal get_TriggerTimestamp ();
```

Arguments:

None.

Exceptions Thrown:

InvalidOperationException :
  Condition: No trigger sample exists.
  Message: No trigger timestamp exists.

Remarks:

The HasTriggerSample property should be checked before accessing this property.

3.13.1.7 AcquisitionSummary.SamplePeriod

Description:

The read-only SamplePeriod property returns the sample period in picoseconds.

Declaration Syntax:

Visual Basic

```
ReadOnly Property SamplePeriod As Decimal
```

C#

```
decimal SamplePeriod { get; }
```

C++

```
Decimal get_SamplePeriod ();
```

Arguments:

None.

Exceptions Thrown:

None.

Remarks:

Not all timebases operate at a fixed sample period or for every acquisition. A sample period of zero indicates that no fixed sample period was in effect for the timebase at the time of the acquisition.

3.14 IRecordAccess

Description:

This interface supports remote access to acquisition data records. Instances of these objects are obtained through a factory interface (IAcquisitionData) by requesting an instrument to create such an object.

Namespace: Tektronix.LogicAnalyzer.TpiNet

Declaration Syntax:

Visual Basic

```
Interface IRecordAccess Inherits IValidity Inherits IDisposable
```

C#

```
interface IRecordAccess : IValidity, IDisposable
```

C++

```
__gc __interface IRecordAccess : public IValidity, public IDisposable
```

Remarks:

The IRecordAccess object may be used to obtain acquisition data across multiple acquisitions.

When the IAcquisitionData object Dispose() method is called, any IRecordAccess objects that it created are also told to Dispose(). The IRecordAccess object should not be used after Dispose(). This automatic disposal by the TLA application occurs when the instrument object is destroyed. This may happen during a system load or when merging or unmerging LA modules. The client should also call IRecordAccess.Dispose() when finished with the object to allow the TLA application to free resources associated with the instance.

3.14.1 IRecordAccess Properties

3.14.1.1 IRecordAccess.Polarize

Description:

This property allows control over applying polarity information to the acquisition record requests. The polarize setting must be set prior to acquisition record requests. The polarization is useful for the main or MagniVu data sets of the LA, but has no effect on DSO data sets or on violation data, compare data, or invalid bits data.

Declaration Syntax:

Visual Basic

```
Property Polarize As Boolean
```

C#

```
bool Polarize { set; get;}
```

C++

```
bool get_Polarize ();
void set_Polarize (bool polarize);
```

Arguments:

polarize – A setting of true means to apply polarity on subsequent record retrieval calls if possible. A setting of false means not to apply polarity on subsequent record retrieval calls.

Exceptions Thrown:

None

Remarks:

The default state of this property should be true.

Examples:

C# Example

```
using System;
using Tektronix.LogicAnalyzer.TpiNet
class TestPolarize
{
  static public void CheckPolarity( IRecordAccess iAccess )
  {
    // Check polarization using raw data from the main data set
    iAccess.DataSet( DataSetValue.Main );
    iAccess.FormatRaw();

    // Get the first 10 main raw data as text with polarity applied
    iAccess.Polarize = true;
    String[] recs = iAccess.GetStringRecords( 0, 10 );

    // Dump the polarize data
    Console.WriteLine("Polarized data…");
    for ( int i = 0; i < recs.GetLength(0); i++ )
    {
      Console.WriteLine("{0}", recs[i]);
    }

    // Get the same records without polarity
    iAccess.Polarize = false;
    recs = iAccess.GetStringRecords( 0, 10 );

    // Dump the non-polarized data
    Console.WriteLine("Non-polarized data…");
    for ( int i = 0; i < recs.GetLength(0); i++ )
    {
      Console.WriteLine("{0}", recs[i]);
    }
  }
}
```

3.14.1.2 IRecordAccess.DataSet

Description:

This property allows control over which data set of the instrument is to be targeted for data retrieval.

Declaration Syntax:

Visual Basic

```
Property DataSet As DataSetValue
```

C#

```
DataSetValue DataSet { set; get;}
```

C++

```
DataSetValue get_DataSet ();
void set_DataSet (DataSetValue target);
```

Arguments:

target – Identifies which data set to use for subsequent record requests.

Exceptions Thrown:

ArgumentException :
Condition: Data access for the given data set is not supported.
Message: Acquisition data for the given data set is not supported.

Remarks:

The default state of this property should be DataSetValue.Main.

Examples:

C# Example
```
using System;
using Tektronix.LogicAnalyzer.TpiNet
class TestDataSetTarget
{
  static public void Dump( IRecordAccess iAccess )
  {
    // Get the first 10 raw violation records as text
    iAccess.DataSet = DataSetValue.Violation;
    iAccess.FormatRaw();

    String[] recs = iAccess.GetStringRecords( 0, 10 );

    // Dump the violation data
    Console.WriteLine("{0} data…", iAccess.DataSet);
    for ( int i = 0; i < recs.GetLength(0); i++ )
    {
      Console.WriteLine("{0}", recs[i]);
    }

    // Now get the first 10 raw MagniVu records as text
    iAccess.DataSet = DataSetValue.MagniVu;
    recs = iAccess.GetStringRecords( 0, 10 );

    // Dump the MagniVu data
    Console.WriteLine("{0} data…", iAccess.DataSet);
    for ( int i = 0; i < recs.GetLength(0); i++ )
    {
      Console.WriteLine("{0}", recs[i]);
    }
  }
}
```

3.14.1.3 IRecordAccess.StringSeparator

Description:

This property is only used by the GetStringRecords() method and is used to control the characters which separates fields in a multiple field output format. When the data format is not composed of multiple fields (groups, channels, timestamp), the StringSeparator is not used.

Declaration Syntax:

Visual Basic

```
Property StringSeparator As String
```

C#

```
string StringSeparator { set; get;}
```

C++

```
String* get_StringSeparator ();
void set_StringSeparator (String* separator);
```

Arguments:

separator – A string used to separate fields within a string returned by GetStringRecords().

Exceptions Thrown:

None

Remarks:

The default state of this property should be a string containing a single space character.

3.14.1.4 IRecordAccess.TimestampGroupName

Description:

This property represents the name of the timestamp group for use with FormatLogical() or FormatObjects().

Declaration Syntax:

Visual Basic

```
ReadOnly Property TimestampGroupName As String
```

C#

```
string TimestampGroupName { get;}
```

C++

```
String* get_TimestampGroupName ();
```

Arguments:

None.

Exceptions Thrown:

None

Remarks:

If a module does not support a timestamp group, this property is the empty string.

3.14.1.5 IRecordAccess.RawTimestampMultiplier

Description:

This property represents a multiplication factor to apply to raw timestamps. Multiplying a timestamp obtained under FormatRaw() mode by this factor yields a timestamp in picoseconds.

Declaration Syntax:

Visual Basic

```
ReadOnly Property RawTimestampMultiplier As Decimal
```

C#

```
decimal RawTimestampMultiplier { get;}
```

C++

```
Decimal get_RawTimestampMultiplier ();
```

Arguments:

None.

Exceptions Thrown:

InvalidOperationException :
Condition: IAcquisitionData.IsValid is false for the object that created this IRecordAccess object.
Message: Module status values are not valid at this time.

Remarks:

Several raw formats do not include timestamp bits in their record format. The RawTimestampMultiplier value only has meaning for raw formats that do contain timestamp bits.

The RawTimestampMultiplier for an LA module is a valid multiplier than can be used on data sets other than MagniVu. Since DSO and Oscilloscope modules never provide timestamp bits under RawFormat(), the RawTimestampMultiplier for these modules is zero.

3.14.2 IRecordAccess Methods

3.14.2.1 IRecordAccess.IsDataSetSupported

Description:

This method is used to see if the instrument supports the designated data set.

Declaration Syntax:

Visual Basic

```
Function Method-Name (Argument As Type-Name) As Type-Name
      - or -
Sub Method-Name (Argument As Type-Name)
```

C#

```
bool IsDataSetSupported ( DataSetValue which )
```

C++

```
bool IsDataSetSupported ( DataSetValue which )
```

Arguments:

which – a data set to check.

Return Value:

A return of true means the instrument supports the designated data set. A return of false means the instrument does not support the designated data set.

Exceptions Thrown:

None

Remarks:

Even though an instrument may return true to indicate support of a data set, it does not follow that acquisition data exists for that data set. To check on acquisition data availability for a data set, use the NumberOfRecords method.

Is this method even necessary? Maybe NumberOfRecords() is sufficient.

Examples:

C# Example
```
using System
using Tektronix.LogicAnalyzer.TpiNet

class TestIsDataSetSupported
{
  // Prints if a couple of data sets are supported by LA/DSO.
  static public void Test( IRecordAccess iAccessLA,
                           IRecordAccess iAccessDSO )
  {
    IRecordAccess[] iAccess = { iAccessLA, iAccessDSO };
    string[] instType = { "LA", "DSO" };
    DataSetValue[] dset = { DataSetValue.Main, DataSetValue.MagniVu };

    for ( int nDset = 0; nDset < dset.Length; nDset++ )
    {
      for ( int nInst = 0; nInst < instType.Length; nInst++ )
      {
        Console.Write( "{0} {1}", instType[nInst], dset[nDset] );
        if ( iAccess[nInst].IsDataSetSupported( dset[nDset] )
        {
          Console.WriteLine( " is supported" );
        }
        else
        {
          Console.WriteLine( " is not supported" );
        }
      }
    }
  }
}
```

3.14.2.2 IRecordAccess.NumberOfRecords

Description:

This method reports how many samples are available for the designated data set.

Declaration Syntax:

Visual Basic

```
Function NumberOfRecords (which As DataSetValue) As Long
```

C#

```
long NumberOfRecords ( DataSetValue which )
```

C++

```
__int64 NumberOfRecords ( DataSetValue which )
```

Arguments:

which – a data set to check.

Return Value:

Returns the number of acquisition records available for the designated data set.

Exceptions Thrown:

None

Remarks:

None.

Examples:

C# Example

```
using System
using Tektronix.LogicAnalyzer.TpiNet

class TestNumberOfRecords
{
  // Prints the number of records available for all data sets
  static public void Test( IRecordAccess iAccess )
  {
    Array dset = Enum.GetValues( typeof(DataSetValue) );
    for ( int i = 0; i < dset.Length; i++ )
    {
      DataSetValue curDataSet = dset[i];
      Console.WriteLine( "Data Set {0}: {1} records",
                         curDataSet,
                         iAccess.NumberOfRecords(curDataSet) );
    }
    }
}
```

3.14.2.3 IRecordAccess.FormatRaw

Description:

This method specifies to return records in raw acquisition format.

Declaration Syntax:

Visual Basic

```
Function Method-Name (Argument As Type-Name) As Type-Name
      - or -
Sub Method-Name (Argument As Type-Name)
```

C#

```
void FormatRaw ()
```

C++

```
void FormatRaw ()
```

Arguments:

None.

Return Value:

None.

Exceptions Thrown:

None.

Remarks:

Raw format returns full length acquisition vectors. The raw format will differ depending on the instrument type (34 channel LA vs. 68 channel LA vs. 2 channel DSO vs. 4 channel DSO). The raw format may also differ depending on the DataSet property. LA main, violation, invalid bits, and compare data includes raw timestamp bits, however only main and violation contain meaningful timestamp bits, the rest are zero filled.

LA MagniVu raw format does not include timestamp bits.

Record formats for different module types are documented in Appendix A of this document.

Examples:

C# Example
```
using System
using Tektronix.LogicAnalyzer.TpiNet

class TestFormatRaw
{
  // This dumps an overview of the data returned by GetByteRecords.
  static public void Test( IRecordAccess iAccess,
                           long start,
                           int count )
  {
    // Specify raw format for subsequent data requests.
    iAccess.FormatRaw();

    // Request the samples
    Console.WriteLine("Requested {0} records starting at {1}",
                      count, start);
    String[] recs = iAccess.GetStringRecords(start, count);

    int numRecs = recs.GetLength(0);
    Console.WriteLine("Obtained {0} records" numRecs(0);

    for ( int i = 0; i < numRecs; i++ )
    {
      Console.WriteLine("{0}", recs[i]);
    }
  }
}
```

3.14.2.4 IRecordAccess.FormatLogical

Description:

This method specifies to return records in a logically grouped format.

Declaration Syntax:

C#

```
void FormatLogical ( String[] groupings )
```

C++

```
void FormatLogical ( String* groupings __gc[] )
```

Arguments:

Groupings – An array of strings. Each string specifies a particular set of bits from the acquisition record. For an LA, a string could specify the name of a channel group, a channel, or the string returned by IRecordAccess.TimestampGroupName. For a DSO, a string could specify a DSO channel.

Return Value:

None.

Exceptions Thrown:

ArgumentException :
  Condition: Invalid groupings are specified.
  Message: Invalid grouping specified for record access.

Remarks:

Data for each logical grouping returned by GetByteRecords() is padded to the nearest byte boundary with zero padding in the unused most significant bits. Data for each logical grouping returned by GetStringRecords() consists of the minimal number of hexadecimal digits that are required to represent the number of bits for the grouping. For example, a 10-bit grouping always consists of 3 hexadecimal digits with GetStringRecords().

Timestamp values are returned in picoseconds. No additional time offsets are applied to timestamps (correlation time offset, system trigger time offset, frame offset, or user adjustable module time offset).

A channel and group can have the same name (“CK0” for example). In this case, the group data is returned rather than the channel data. FormatObjects() can be used to avoid group versus channel name ambiguity.

Examples:

C# Example
```
using System
using Tektronix.LogicAnalyzer.TpiNet
```

```
class TestFormatLogical
{
  // This outputs LA data for “A3”, “A2”, and “CK0” at sample zero
  // and outputs DSO data for “Ch1” and “Ch2[0]” at sample zero.
  static public void Test( IRecordAccess iAccessLA,
                           IRecordAccess iAccessDSO )
  {
    // LA format includes two sections and a channel
    String[] formatLA = {“A3”, “A2”, “CK0”};
    iAccessLA.FormatLogical( formatLA );
    iAccessLA.StringSeparator = ‘:‘ ;

    // DSO format includes a channel and the LSBit of another channel
    string[] formatDSO = {“Ch1”, “Ch2[0]”};
    iAccessDSO.FormatLogical( formatDSO );
    iAccessDSO.StringSeparator = ‘:‘;

    // Output for LA
    String[] recsLA = iAccessLA.GetStringRecords(0, 1);
    Console.WriteLine(“LA output should be 2+1+2+1+1->7 characters…”);
    if ( recsLA.GetLength(0) > 0 )
    {
      Console.WriteLine(“{0}”, recsLA[0]);
    }

    // Output for DSO
    String[] recsDSO = iAccessDSO.GetStringRecords(0, 1);
    Console.WriteLine(“DSO output should be 4+1+1->6 characters…”);
    if ( recsDSO.GetLength(0) > 0 )
    {
      Console.WriteLine(“{0}”, recsDSO[0]);
    }
}
```

3.14.2.5 IRecordAccess.FormatObjects

Description:

This method specifies to return records for the designated objects.

Declaration Syntax:

C#

```
void FormatObjects ( object[] groupings )
```

C++

```
void FormatObjects ( Object* groupings __gc[] )
```

Arguments:

Groupings – An array of object. Each object is either an IChannelGroup, IChannelBit, or the string returned by IRecordAccess.TimestampGroupName.

Return Value:

None.

Exceptions Thrown:

ArgumentException :

Condition: Invalid groupings are specified.

Message: Invalid grouping specified for record access.

Remarks:

Data for each logical grouping returned by GetByteRecords() is padded to the nearest byte boundary with zero padding in the unused most significant bits. Data for each logical grouping returned by GetStringRecords() consists of the minimal number of hexadecimal digits that are required to represent the number of bits for the grouping. For example, a 10-bit grouping always consists of 3 hexadecimal digits with GetStringRecords().

Timestamp values are returned in picoseconds. No additional time offsets are applied to timestamps (correlation time offset, system trigger time offset, frame offset, or user adjustable module time offset).

Examples:

C# Example
```
using System
using Tektronix.LogicAnalyzer.TpiNet

class TestFormatObjects
{
  // This outputs LA data for “A3”, “A2”, and “CK0” at sample zero
  static public void Test( IRecordAccess iAccessLA, ILAModule module )
  {
    // LA default grouping includes two sections and a channel
    ArrayList groupingList = module.GetChannelGroupingObject().Groups;
    object[] myObjs = groupingList.ToArray();
    iAccessLA.FormatObjects( myObjs );
    iAccessLA.StringSeparator = ‘:‘ ;

    // Output for LA
    String[] recsLA = iAccessLA.GetStringRecords(0, 1);
    Console.WriteLine(“LA output should be 2+1+2+1+1->7 characters…”);
    if ( recsLA.GetLength(0) > 0 )
    {
      Console.WriteLine(“{0}”, recsLA[0]);
    }
}
```

3.14.2.6 IRecordAccess.GetByteRecords

Description:

This method returns multiple binary acquisition records in a two-dimensional byte array.

Declaration Syntax:

C#

```
System.Array GetByteRecords ([in]long start, [in]int count)
```

C++

```
System::Array* GetByteRecords ([in]__int64 start, [in]int count)
__gc[,]
```

Arguments:

start – This specifies the first record number to retrieve.
count – This specifies how many records to retrieve.

Return Value:

The number of samples returned are represented by the return array GetLength[0]. It is possible for fewer than the number of requested samples to be returned, so GetLength[0] should always be checked. The number of bytes per sample may be obtained with GetLength[1].

Exceptions Thrown:

ArgumentOutOfRangeException :
Condition: The number of samples acquired does not include the specified start sample.
Message: Acquisition record requested is out of range.

Remarks:

None.

Examples:

C# Example
```
using System
using Tektronix.LogicAnalyzer.TpiNet

class TestGetByteRecords
{
  // This dumps an overview of the data returned by GetByteRecords.
  static public void Test( IRecordAccess iAccess,
                           long start,
                           int count )
  {
    // Request the samples
    Console.WriteLine("Requested {0} records starting at {1}",
                      count, start);
    byte[,] recs = (byte[,]) iAccess.GetByteRecords(start, count);
    int numRecs = recs.GetLength(0);
    Console.WriteLine("Obtained {0} records" numRecs(0);

    if (numRecs <= 0)
    {
      Console.WriteLine("No data available");
      return;
    }
    if (numRecs != count)
    {
      Console.WriteLine("Number of records returned differs from
request");
    }

    int lengthPerRec = recs.GetLength(1);

    Console.WriteLine("Last record is {0}", start + numRecs - 1);
    Console.WriteLine("Length per record is {0}", lengthPerRec);
    Console.WriteLine("Last byte of sample {0} is {1:H}",
                      start, recs[0,(lengthPerRec-1)]);
  }
}
```

3.14.2.7 IRecordAccess.GetStringRecords

Description:

This method returns multiple acquisition records in a one-dimensional String array.

Declaration Syntax:

C#

```
String[] GetStringRecords ([in]long start, [in]int count)
```

C++

```
String* GetStringRecords ([in]__int64 start, [in]int count) __gc[]
```

Arguments:

start – This specifies the first record number to retrieve.
count – This specifies how many records to retrieve.

Return Value:

The number of samples returned are represented by the return array GetLength[0]. It is possible for fewer than the number of requested samples to be returned, so GetLength[0] should always be checked.

Exceptions Thrown:

ArgumentOutOfRangeException :
Condition: The number of samples acquired does not include the specified start sample.
Message: Acquisition record requested is out of range.

Remarks:

None.

Examples:

C# Example
```
using System
using Tektronix.LogicAnalyzer.TpiNet

class TestGetStringRecords
{
  // This dumps the data returned by GetStringRecords.
  static public void Test( IRecordAccess iAccess,
                           long start,
                           int count )
  {
    // Request the samples
    Console.WriteLine("Requested {0} records starting at {1}",
                       count, start);
    byte[,] recs = iAccess.GetStringRecords(start, count);
    int numRecs = recs.GetLength(0);
    Console.WriteLine("Obtained {0} records" numRecs(0);

    if (numRecs <= 0)
    {
      Console.WriteLine("No data available");
      return;
    }
    if (numRecs != count)
    {
      Console.WriteLine("Number of records returned differs from
request");
    }

    // Dump all records
    for (int i = 0; i < numRecs; i++)
    {
      String sData = recs[i];
      Console.WriteLine("Rec {0}: {1}", start+i, sData);
    }
  }
}
```

3.15 IDataSourceTrigger

Description:

This is the base interface for all data source trigger definitions, including Logic analyzer, DSO, and plug-in trigger definitions. IDataSourceTrigger contains only those members that must be implemented by all data sources with triggering.

Namespace: Tektronix.LogicAnalyzer.Common

Declaration Syntax:

Visual Basic

```
Interface IDataSourceTrigger
      Inherits IValidity
      Inherits IDefault
      Inherits IDisposable
```

C#

```
interface IDataSourceTrigger
      : IValidity, IDefault, IDisposable
```

C++

```
__gc __interface IDataSourceTrigger
      : public IValidity, IDefault, IDisposable
```

Remarks:

The purpose of this interface is to allow methods that deal with trigger properties common to all kinds of data sources.

3.15.1 IDataSourceTrigger Properties

3.15.1.1 IDataSourceTrigger.Position

Description:

The percentage offset, from the beginning of acquired data, to store the data sample associated with the data source’s trigger event.

Declaration Syntax:

Visual Basic

```
Property Position As Long
```

C#

```
long Position { set; get; }
```

C++

```
__int64 get_Position ();
void set_Position (__int64 position );
```

Arguments:

position – a value between 0 and 100 that specifies the percentage offset from the beginning of acquired data at which the trigger-event sample should be stored.

Exceptions Thrown:

ArgumentOutOfRangeException:
    Condition: Trigger position percentage is out of bounds.
    Message: The requested trigger position percentage is outside the range currently possible.

Remarks:

The available range of trigger positions, expressed as a percentage of acquired memory samples, may lie somewhere within 0 to 100%. Query the IDataSourceTrigger.MinimumPosition and IDataSourceTrigger.MaximumPosition properties to determine the currently valid range.

Examples:

```
See examples for ILATrigger.
```

3.15.1.2 IDataSourceTrigger.MinimumPosition

Description:

The minimum percentage offset, from the beginning of acquired data, that the data sample associated with the data source's trigger event can be stored.

Declaration Syntax:

Visual Basic

```
ReadOnly Property MinimumPosition As Long
```

C#

```
long MinimumPosition { get; }
```

C++

```
__int64 get_MinimumPosition ();
```

Arguments:

None.

Exceptions Thrown:

None.

Remarks:

The minimum available trigger position, expressed as a percentage of acquired memory samples, may be greater than 0%.

Examples:

```
See examples for ILATrigger.
```

3.15.1.3 IDataSourceTrigger.MaximumPosition

Description:

The maximum percentage offset, from the beginning of acquired data, that the data sample associated with the data source's trigger event can be stored.

Declaration Syntax:

Visual Basic

```
ReadOnly Property MaximumPosition As Long
```

C#

```
long MaximumPosition { get; }
```

C++

```
__int64 get_MaximumPosition ();
```

Arguments:

None.

Exceptions Thrown:

None.

Remarks:

The maximum available trigger position, expressed as a percentage of acquired memory samples, may be less than 100%.

Examples:

```
See examples for ILATrigger.
```

3.15.1.4 IDataSourceTrigger.NumberOfStates

Description:

Returns the number of states currently in the trigger definition.

Declaration Syntax:

Visual Basic

```
ReadOnly Property NumberOfStates As Int32
```

C#

```
Int32 NumberOfStates { get; }
```

C++

```
Int32 get_NumberOfStates ()
```

Arguments:

None.

Exceptions Thrown:

None.

Remarks:

None.

Examples:

```
See examples for ILATrigger.
```

3.15.2 IDataSourceTrigger Methods

3.15.2.1 IDataSourceTrigger.CanAddState

Description:

Indicates whether a new state can be added to the trigger definition.

Declaration Syntax:

Visual Basic

```
Function CanAddState () As Boolean
Function CanAddState ( state As IDataSourceTriggerState ) As Boolean
```

C#

```
bool CanAddState ()
bool CanAddState ( IDataSourceTriggerState state )
```

C++

```
bool CanAddState ()
bool CanAddState ( IDataSourceTriggerState* state )
```

Arguments:

`state` – the pre-initialized state to be added.

Return Value:

Value of true indicates that a new state can be added to the trigger definition.

Exceptions Thrown:

None.

Remarks:

The first form of the function indicates whether a default state can be added to the trigger definition.

The second form indicates whether the specified state can be added. In general, a non-default state is more likely to encounter resource allocation conflicts that prevent it from being added to the trigger definition.

The second for also uses reflection to determine whether the state interface is compatible with the current trigger interface.

Examples:

```
See examples for ILATrigger.
```

3.15.2.2 IDataSourceTrigger.AddState

Description:

Appends a new state to the trigger definition.

Declaration Syntax:

Visual Basic

```
Sub AddState ( state as IDataSourceTriggerState )
```

C#

```
void AddState ( IDataSourceTriggerState state )
```

C++

```
void AddState (IDataSourceTriggerState* state )
```

Arguments:

`state` – the pre-initialized state to be added.

Return Value:

None.

Exceptions Thrown:

ArgumentOutOfRangeException::
Condition: Out-of-bounds state index.
Message: The state index does not represent a valid trigger state location.

TlaTriggerResourceException:
Condition: An event in the trigger state conflicts with the current trigger definition.
Message: The state cannot be added because at least one of its events conflicts with the current trigger definition.

TlaTriggerResourceException:
Condition: An action in the trigger state conflicts with the current trigger definition.
Message: The state cannot be added because at least one of its actions conflicts with the current trigger definition.

TlaTriggerResourceException:
Condition: The trigger definition already has the maximum number of states possible.
Message: The state cannot be added because the trigger definition already has the maximum number of states possible.

Remarks:

None.

Examples:

```
See examples for ILATrigger.
```

3.15.2.3 IDataSourceTrigger.InsertState

Description:

Adds a new state to the trigger definition at the specified location.

Declaration Syntax:

Visual Basic

```
Sub InsertState ( index As Int32,
                  state as IDataSourceTriggerState )
```

C#

```
void InsertState ( Int32 index,
                   IDataSourceTriggerState state )
```

C++

```
void InsertState ( Int32 index,
                   IDataSourceTriggerState* state )
```

Arguments:

`index` – zero-based index at which the new trigger state should be added.

`state` – the pre-initialized state to be added.

Return Value:

None.

Exceptions Thrown:

ArgumentOutOfRangeException:
Condition: Out-of-bounds state index.
Message: The state index does not represent a valid trigger state location.

ArgumentException:
Condition: Incomplete state interface.
Message: The specified state interface doesn't support all required functionality.

TlaTriggerResourceException:
Condition: An event in the trigger state conflicts with the current trigger definition.
Message: The state cannot be added because at least one of its events conflicts with the current trigger definition.

TlaTriggerResourceException:
Condition: An action in the trigger state conflicts with the current trigger definition.
Message: The state cannot be added because at least one of its actions conflicts with the current trigger definition.

TlaTriggerResourceException:

Condition: The trigger definition already has the maximum number of states possible.

Message: The state cannot be added because the trigger definition already has the maximum number of states possible.

Remarks:

None.

Examples:

```
See examples for ILATrigger.
```

3.15.2.4 IDataSourceTrigger.RemoveState

Description:

Removes the specified state from the trigger definition.

Declaration Syntax:

Visual Basic

```
Sub RemoveState (index As Int32 )
```

C#

```
void RemoveState ( Int32 index)
```

C++

```
void RemoveState ( Int32 index)
```

Arguments:

`index` – zero-based index of the trigger state to remove.

Return Value:

None.

Exceptions Thrown:

ArgumentOutOfRangeException:
Condition: Out-of-bounds state index.
Message: The state index does not represent a valid trigger state location.

TlaTriggerResourceException:
Condition: Only one state remains in the trigger definition. It can’t be removed.
Message: The last state in the trigger definition cannot be removed.

Remarks:

None.

Examples:

```
See examples for ILATrigger.
```

3.15.2.5 IDataSourceTrigger.GetState

Description:

Returns the specified state from the trigger definition.

Declaration Syntax:

Visual Basic

```
Function GetState (index As Int32 ) As IDataSourceTriggerState
```

C#

```
IDataSourceTriggerState GetState ( Int32 index )
```

C++

```
IDataSourceTriggerState * GetState ( Int32 index )
```

Arguments:

`index` – zero-based index of the trigger state to return.

Return Value:

The interface to the specified trigger state.

Exceptions Thrown:

ArgumentOutOfRangeException:
    Condition: Out-of-bounds state index.
    Message: The state index does not represent a valid trigger state location.

Remarks:

None.

Examples:

```
See examples for ILATrigger.
```

3.15.3 IDataSourceTrigger Events

3.15.3.1 IDataSourceTrigger.PositionChanged

Description:

When the trigger position changes, the PositionChanged event is raised.

Declaration Syntax:

Visual Basic

```
Event PositionChanged As ObjectEventHandler
```

C#

```
event ObjectEventHandler PositionChanged;
```

C++

```
__event  ObjectEventHandler PositionChanged;
```

Event Data:

The IDataSourceTrigger object is passed as a parameter.

Remarks:

None.

Examples:

```
See examples for ILATrigger.
```

3.15.3.2 IDataSourceTrigger.NumberOfStatesChanged

Description:

When a trigger state is added or deleted, the NumberOfStatesChanged event is raised.

Declaration Syntax:

Visual Basic

```
Event NumberOfStatesChanged As ObjectEventHandler
```

C#

```
event ObjectEventHandler NumberOfStatesChanged;
```

C++

```
__event  ObjectEventHandler NumberOfStatesChanged;
```

Event Data:

The IDataSourceTrigger object is passed as a parameter.

Remarks:

None.

Examples:

```
See examples for ILATrigger.
```

3.16 IDataSourceTriggerState

Description:

This is the base interface for all data source trigger states, including Logic analyzer, DSO, and plug-in trigger definitions. IDataSourceTriggerState contains only those members that must be implemented by all data sources.

Namespace: Tektronix.LogicAnalyzer.Common

Declaration Syntax:

Visual Basic

```
Interface IDataSourceTriggerState
        Inherits IValidity
        Inherits IDefault
        Inherits IDisposable
        Inherits ICloneable
```

C#

```
interface IdataSourceTriggerState
        : IValidity, IDefault, IDisposable, ICloneable
```

C++

```
__gc __interface IdataSourceTriggerState
        : public IValidity, IDefault, IDisposable, ICloneable
```

Remarks:

The purpose of this interface is to allow methods that deal with trigger state properties common to all kinds of data sources.

3.16.1 IDataSourceTriggerState Properties

3.16.1.1 IDataSourceTriggerState.NumberOfClauses

Description:

Returns the number of clauses defined in the trigger state.

Declaration Syntax:

Visual Basic

```
ReadOnly Property NumberOfClauses As Int32
```

C#

```
Int32 NumberOfClauses { get; }
```

C++

```
Int32 get_NumberOfClauses ()
```

Arguments:

None.

Exceptions Thrown:

None.

Remarks:

None.

Examples:

```
See examples for ILATriggerState.
```

3.16.2 IDataSourceTriggerState Methods

3.16.2.1 IDataSourceTriggerState.CanAddClause

Description:

Indicates whether a new clause can be added to the trigger state.

Declaration Syntax:

Visual Basic

```
Function CanAddClause () As Boolean
Function CanAddClause ( clause As IDataSourceTriggerClause ) As Boolean
```

C#

```
bool CanAddClause ()
bool CanAddClause ( IDataSourceTriggerClause clause )
```

C++

```
bool CanAddClause ()
bool CanAddClause ( IDataSourceTriggerClause* clause )
```

Arguments:

`clause` – a pre-defined trigger clause.

Return Value:

Value of true indicates that a new clause can be added to the trigger state.

Exceptions Thrown:

None.

Remarks:

The first version of the method indicates whether a default trigger clause can be added to the trigger state.

The second version of the method indicates whether the specified clause can be added to the trigger state. In general a non-default clause is more likely to have resource allocation conflicts that prevent it from being added to a state.

The second method also utilizes reflection to determine that the IDataSourceTriggerClause-derived interface is compatible with the trigger state.

The number of clauses a state can have depends on the number of events already in use within the state, how those events are combined within the state's existing clauses, and the number and combination of events in the clause to be added.

Examples:

```
See examples for ILATriggerState.
```

3.16.2.2 IDataSourceTriggerState.AddClause

Description:

Appends a new clause to the trigger state.

Declaration Syntax:

Visual Basic

```
Sub AddClause (clause as IDataSourceTriggerClause )
```

C#

```
void AddClause ( IDataSourceTriggerClause clause )
```

C++

```
void AddClause ( IDataSourceTriggerClause * clause )
```

Arguments:

`clause` – the pre-initialized clause to be added.

Return Value:

None.

Exceptions Thrown:

ArgumentException:
Condition: Incomplete clause interface.
Message: The specified clause interface is does not support all required functionality.

TlaTriggerResourceException:
Condition: An event in the trigger clause conflicts with the current trigger definition.
Message: The clause cannot be added because at least one of its events conflicts with the current trigger definition.

TlaTriggerResourceException:
Condition: An action in the trigger clause conflicts with the current trigger definition.
Message: The clause cannot be added because at least one of its actions conflicts with the current trigger definition.

TlaTriggerResourceException:
Condition: The trigger state already has the maximum number of clause possible.
Message: The clause cannot be added because the trigger state already has the maximum number of clauses possible.

Remarks:

None.

Examples:

See examples for ILATriggerState.

3.16.2.3 IDataSourceTriggerState.InsertClause

Description:

Adds a new clause to the trigger state after the specified location.

Declaration Syntax:

Visual Basic

```
Sub InsertClause ( index As Int32,
                   clause as IDataSourceTriggerClause)
```

C#

```
void InsertClause ( Int32 index,
                    IDataSourceTriggerClause clause)
```

C++

```
void InsertClause ( Int32 index,
                    IDataSourceTriggerClause * clause )
```

Arguments:

`index` – zero-based index of the in the trigger state at which the new clause should be inserted.

`clause` – the pre-initialized clause to be added.

Return Value:

None.

Exceptions Thrown:

ArgumentOutOfRangeException:
    Condition: Out-of-bounds clause index.
    Message: The clause index does not represent a valid location in the trigger state.

ArgumentException:
    Condition: Incomplete clause interface.
    Message: The specified clause interface is does not support all required functionality.

TlaTriggerResourceException:
    Condition: An event in the trigger clause conflicts with the current trigger definition.
    Message: The clause cannot be added because at least one of its events conflicts with the current trigger definition.

TlaTriggerResourceException:
    Condition: An action in the trigger clause conflicts with the current trigger definition.
    Message: The clause cannot be added because at least one of its actions conflicts with the current trigger definition.

TlaTriggerResourceException:

Condition: The trigger state already has the maximum number of clauses possible.

Message: The clause cannot be added because the trigger state already has the maximum number of clauses possible.

Remarks:

None.

Examples:

```
See examples for ILATriggerState.
```

3.16.2.4 IDataSourceTriggerState.RemoveClause

Description:

Removes the specified clause from the trigger state.

Declaration Syntax:

Visual Basic

```
Sub RemoveClause ( index As Int32 )
```

C#

```
void RemoveClause ( Int32 index)
```

C++

```
void RemoveClause ( Int32 index)
```

Arguments:

`index` – zero-based index of the trigger clause to remove.

Return Value:

None.

Exceptions Thrown:

ArgumentOutOfRangeException:
Condition: Out-of-bounds clause index.
Message: The clause index does not represent a valid location in the trigger state.

TlaTriggerResourceException:
Condition: Only one clause remains in the trigger state. It can’t be removed.
Message: The last clause in the trigger state cannot be removed.

Remarks:

None.

Examples:

```
See examples for ILATriggerState.
```

3.16.2.5 IDataSourceTriggerState.GetClause

Description:

Returns the specified clause from the trigger state.

Declaration Syntax:

Visual Basic

```
Function GetClause ( index As Int32 )
      As IDataSourceTriggerClause
```

C#

```
IDataSourceTriggerClause GetClause ( Int32 index )
```

C++

```
IDataSourceTriggerClause * GetClause ( Int32 index )
```

Arguments:

`index` – zero-based index of the trigger clause to return.

Return Value:

The interface to the specified trigger clause.

Exceptions Thrown:

ArgumentOutOfRangeException:
  Condition: Out-of-bounds clause index.
  Message: The clause index does not represent a valid location in the trigger state.

Remarks:

None.

Examples:

```
See examples for ILATriggerState.
```

3.16.3 IDataSourceTriggerState Events

3.16.3.1 IDataSourceTriggerState.NumberOfClausesChanged

Description:

When a trigger clause is added or deleted, the NumberOfClausesChanged event is raised.

Declaration Syntax:

Visual Basic

```
Event NumberOfClausesChanged As ObjectEventHandler
```

C#

```
event ObjectEventHandler NumberOfClausesChanged;
```

C++

```
__event  ObjectEventHandler NumberOfClausesChanged;
```

Event Data:

The IDataSourceTriggerState object is passed as a parameter.

Remarks:

None.

Examples:

```
See examples for ILATriggerState.
```

3.17 IDataSourceTriggerClause

Description:

This is the base interface for all data source trigger clauses, including Logic analyzer, DSO, and plug-in trigger definitions. IDataSourceTriggerClause contains only those members that must be implemented by all data sources.

Namespace: Tektronix.LogicAnalyzer.Common

Declaration Syntax:

Visual Basic

```
Interface IDataSourceTriggerClause
      Inherits IValidity
      Inherits IDefault
      Inherits IDisposable
      Inherits ICloneable
```

C#

```
interface IdataSourceTriggerClause
      : IValidity, IDefault, IDisposable, ICloneable
```

C++

```
__gc __interface IdataSourceTriggerClause
      : public IValidity, IDefault, IDisposable, ICloneable
```

Remarks:

The purpose of this interface is to allow methods that deal with trigger clause properties common to all kinds of data sources.

3.17.1 IDataSourceTriggerClause Properties

3.17.1.1 IDataSourceTriggerClause.NumberOfEvents

Description:

Returns the number of events defined in the trigger clause.

Declaration Syntax:

Visual Basic

```
ReadOnly Property NumberOfEvents As Int32
```

C#

```
Int32 NumberOfEvents { get; }
```

C++

```
Int32 get_NumberOfEvents ()
```

Arguments:

None.

Exceptions Thrown:

None.

Remarks:

None.

Examples:

```
See examples for ILATriggerClause.
```

3.17.1.2 IDataSourceTriggerClause.NumberOfActions

Description:

Returns the number of actions defined in the trigger clause.

Declaration Syntax:

Visual Basic

```
ReadOnly Property NumberOfActions As Int32
```

C#

```
Int32 NumberOfActions { get; }
```

C++

```
Int32 get_NumberOfActions ()
```

Arguments:

None.

Exceptions Thrown:

None.

Remarks:

None.

Examples:

```
See examples for ILATriggerClause.
```

3.17.2 IDataSourceTriggerClause Methods

3.17.2.1 IDataSourceTriggerClause.CanAddEvent

Description:

Indicates whether a new event can be added to the trigger clause.

Declaration Syntax:

Visual Basic

```
Function CanAddEvent () As Boolean
Function CanAddEvent ( event As IDataSourceTriggerEvent ) As Boolean
```

C#

```
bool CanAddEvent ()
bool CanAddEvent ( IDataSourceTriggerEvent event )
```

C++

```
bool CanAddEvent ()
bool CanAddEvent ( IDataSourceTriggerEvent * event )
```

Arguments:

`event` – a pre-defined trigger event.

Return Value:

Value of true indicates that a new event can be added to the trigger clause.

Exceptions Thrown:

None.

Remarks:

The first version of the method indicates whether a default trigger event can be added to the trigger clause.

The second version of the method indicates whether the specified event can be added to the trigger clause. In general a non-default event is more likely to have resource allocation conflicts that prevent it from being added to a clause.

The number of events a clause can have depends on the number of events already in use within the state containing the clause, how those events are combined within the state's existing clauses, and the number and combination of events in the clause to which the event is to be added.

The second version of the method also utilizes reflection to determine wither the specified event interface is compatible with the current trigger clause interface.

Examples:

`See examples for ILATriggerClause.`

3.17.2.2 IDataSourceTriggerClause.AddEvent

Description:

Appends a new event to the trigger clause.

Declaration Syntax:

Visual Basic

```
Sub AddEvent ( event as IDataSourceTriggerEvent )
```

C#

```
void AddEvent ( IDataSourceTriggerEvent event )
```

C++

```
void AddEvent ( IDataSourceTriggerEvent * event )
```

Arguments:

`event` – the pre-initialized event to be added.

Return Value:

None.

Exceptions Thrown:

ArgumentException:
Condition: Incomplete event interface.
Message: The specified event interface does not support all necessary functionality.

TlaTriggerResourceException:
Condition: The specified event conflicts with the current trigger definition.
Message: The event cannot be added because it conflicts with the current trigger definition.

TlaTriggerResourceException:
Condition: The trigger clause already has the maximum number of events possible.
Message: The event cannot be added because the trigger clause already has the maximum number of events possible.

Remarks:

None.

Examples:

```
See examples for ILATriggerClause.
```

3.17.2.3 IDataSourceTriggerClause.InsertEvent

Description:

Inserts a new event to the trigger clause at the specified location.

Declaration Syntax:

Visual Basic

```
Sub InsertEvent ( index As Integer,
                  event as IDataSourceTriggerEvent )
```

C#

```
void InsertEvent ( int index,
                   IDataSourceTriggerEvent event )
```

C++

```
void InsertEvent ( int index,
                   IDataSourceTriggerEvent * event )
```

Arguments:

`index` – zero-based index of the location in the trigger clause at which to add the new event.

`event` – the pre-initialized event to be added.

Return Value:

None.

Exceptions Thrown:

ArgumentOutOfRangeException:
Condition: Out-of-bounds event index.
Message: The event index does not represent a valid location in the trigger clause.

ArgumentException:
Condition: Incomplete event interface.
Message: The specified event interface does not support all necessary functionality.

TlaTriggerResourceException:
Condition: The specified event conflicts with the current trigger definition.
Message: The event cannot be added because it conflicts with the current trigger definition.

TlaTriggerResourceException:
Condition: The trigger clause already has the maximum number of events possible.
Message: The event cannot be added because the trigger clause already has the maximum number of events possible.

Remarks:

None.

Examples:

```
See examples for ILATriggerClause.
```

3.17.2.4 IDataSourceTriggerClause.RemoveEvent

Description:

Removes the specified event from the trigger clause.

Declaration Syntax:

Visual Basic

```
Sub RemoveEvent ( index As Integer )
```

C#

```
void RemoveEvent ( int index )
```

C++

```
void RemoveEvent ( int index )
```

Arguments:

`index` – zero-based index of the event to remove.

Return Value:

None.

Exceptions Thrown:

ArgumentOutOfRangeException:
Condition: Out-of-bounds event index.
Message: The event index does not represent a valid location in the trigger clause.

TlaTriggerResourceException:
Condition: Only one event remains in the trigger clause. It can't be removed.
Message: The last event in the trigger clause cannot be removed.

Remarks:

None.

Examples:

```
See examples for ILATriggerClause.
```

3.17.2.5 IDataSourceTriggerClause.GetEvent

Description:

Returns the specified event from the trigger clause.

Declaration Syntax:

Visual Basic

```
Function GetEvent ( index As Integer )
      As IDataSourceTriggerEvent
```

C#

```
IDataSourceTriggerEvent GetEvent ( int index )
```

C++

```
IDataSourceTriggerEvent * GetEvent ( int index )
```

Arguments:

index – zero-based index of the trigger event to return.

Return Value:

The interface to the specified trigger event.

Exceptions Thrown:

ArgumentOutOfRangeException:
Condition: Out-of-bounds event index.
Message: The event index does not represent a valid location in the trigger clause.

Remarks:

None.

Examples:

```
See examples for ILATriggerClause.
```

3.17.2.6 IDataSourceTriggerClause.CanAddAction

Description:

Indicates whether a new action can be added to the trigger clause.

Declaration Syntax:

Visual Basic

```
Function CanAddAction () As Boolean
Function CanAddAction ( action As IDataSourceTriggerAction ) As Boolean
```

C#

```
bool CanAddAction ()
bool CanAddAction ( IDataSourceTriggerAction action )
```

C++

```
bool CanAddAction ()
bool CanAddAction ( IDataSourceTriggerAction * action )
```

Arguments:

`action` – a pre-defined trigger action.

Return Value:

Value of true indicates that a new action can be added to the trigger clause.

Exceptions Thrown:

None.

Remarks:

The first version of the method indicates whether a default trigger action can be added to the trigger clause.

The second version of the method indicates whether the specified action can be added to the trigger clause. In general a non-default action is more likely to have resource allocation conflicts that prevent it from being added to a clause.

The number of actions a clause can have depends on the number of actions already in use within the state containing the clause, how those actions are combined within the state's existing clauses, and the number and combination of actions in the clause to which the actions is to be added.

The second version of the method also utilizes reflection to determine whether the specified action interface is compatible with the current trigger clause interface.

Examples:

See examples for ILATriggerClause.

3.17.2.7 IDataSourceTriggerClause.AddAction

Description:

Appends a new action to the trigger clause.

Declaration Syntax:

Visual Basic

```
Sub AddAction ( action as IDataSourceTriggerAction )
```

C#

```
void AddAction ( IDataSourceTriggerAction action )
```

C++

```
void AddAction ( IDataSourceTriggerAction * action )
```

Arguments:

`action` – the pre-initialized action to be added.

Return Value:

None.

Exceptions Thrown:

ArgumentException:
Condition: Incomplete action interface.
Message: The specified action interface does not support all required functionality.

TlaTriggerResourceException:
Condition: The specified action conflicts with the current trigger definition.
Message: The action cannot be added because it conflicts with the current trigger definition.

TlaTriggerResourceException:
Condition: The trigger clause already has the maximum number of action possible.
Message: The action cannot be added because the trigger clause already has the maximum number of actions possible.

Remarks:

None.

Examples:

```
See examples for ILATriggerClause.
```

3.17.2.8 IDataSourceTriggerClause.InsertAction

Description:

Inserts a new action into the trigger clause at the specified location

Declaration Syntax:

Visual Basic

```
Sub AddActionAfter ( index As Integer,
                     action as IDataSourceTriggerAction )
```

C#

```
void AddActionAfter ( int index,
                      IDataSourceTriggerAction action )
```

C++

```
void AddActionAfter ( int index,
                      IDataSourceTriggerAction * action )
```

Arguments:

`index` – zero-based index of the location in the trigger clause at which to insert the new action.

`action` – the pre-initialized action to be added.

Return Value:

None.

Exceptions Thrown:

ArgumentOutOfRangeException:
Condition: Out-of-bounds action index.
Message: The action index does not represent a valid location in the trigger clause.

ArgumentException:
Condition: Incomplete action interface.
Message: The specified action interface does not support all required functionality.

TlaTriggerResourceException:
Condition: The specified action conflicts with the current trigger definition.
Message: The action cannot be added because it conflicts with the current trigger definition.

TlaTriggerResourceException:
Condition: The trigger clause already has the maximum number of actions possible.
Message: The action cannot be added because the trigger clause already has the maximum number of actions possible.

Remarks:

None.

Examples:

```
See examples for ILATriggerClause.
```

3.17.2.9 IDataSourceTriggerClause.RemoveAction

Description:

Removes the specified action from the trigger clause.

Declaration Syntax:

Visual Basic

```
Sub RemoveAction ( index As Integer )
```

C#

```
void RemoveAction ( int index )
```

C++

```
void RemoveAction ( int index )
```

Arguments:

`index` – zero-based index of the action to remove.

Return Value:

None.

Exceptions Thrown:

ArgumentOutOfRangeException:
    Condition: Out-of-bounds action index.
    Message: The action index does not represent a valid location in the trigger clause.

TlaTriggerResourceException:
    Condition: Only one action remains in the trigger clause. It can't be removed.
    Message: The last action in the trigger clause cannot be removed.

Remarks:

None.

Examples:

```
See examples for ILATriggerClause.
```

3.17.2.10 IDataSourceTriggerClause.GetAction

Description:

Returns the specified action from the trigger clause.

Declaration Syntax:

Visual Basic

```
Function GetAction ( index As Integer )
      As IDataSourceTriggerAction
```

C#

```
IDataSourceTriggerAction GetAction ( int index )
```

C++

```
IDataSourceTriggerAction * GetAction ( int index )
```

Arguments:

`index` – zero-based index of the trigger action to return.

Return Value:

The interface to the specified trigger action.

Exceptions Thrown:

ArgumentOutOfRangeException:
    Condition: Out-of-bounds action index.
    Message: The action index does not represent a valid location in the trigger clause.

Remarks:

None.

Examples:

```
See examples for ILATriggerClause.
```

3.17.3 IDataSourceTriggerClause Events

3.17.3.1 IDataSourceTriggerClause.NumberOfEventsChanged

Description:

When a trigger event is added or deleted, the NumberOfEventsChanged event is raised.

Declaration Syntax:

Visual Basic

```
Event NumberOfEventsChanged As ObjectEventHandler
```

C#

```
event ObjectEventHandler NumberOfEventsChanged;
```

C++

```
__event  ObjectEventHandler NumberOfEventsChanged;
```

Event Data:

The IDataSourceTriggerClause object is passed as a parameter.

Remarks:

None.

Examples:

```
See examples for ILATriggerClause.
```

3.17.3.2 IDataSourceTriggerClause.NumberOfActionsChanged

Description:

When a trigger event is added or deleted, the NumberOfActionsChanged event is raised.

Declaration Syntax:

Visual Basic

```
Event NumberOfActionsChanged As ObjectEventHandler
```

C#

```
event ObjectEventHandler NumberOfActionsChanged;
```

C++

```
__event  ObjectEventHandler NumberOfActionsChanged;
```

Event Data:

The IDataSourceTriggerClause object is passed as a parameter.

Remarks:

None.

Examples:

```
See examples for ILATriggerClause.
```

3.18 IDataSourceTriggerEvent

Description:

This is the abstract (pure virtual) base interface for all data source trigger events, including those for Logic analyzer, DSO, and plug-in trigger definitions. IDataSourceTriggerEvent contains only those members that must be implemented by all data sources.

Namespace: Tektronix.LogicAnalyzer.Common

Declaration Syntax:

Visual Basic

```
Interface IDataSourceTriggerEvent
        Inherits IValidity
        Inherits IDefault
        Inherits IDisposable
        Inherits ICloneable
```

C#

```
interface IdataSourceTriggerEvent
        : IValidity, IDefault, IDisposable, ICloneable
```

C++

```
__gc __interface IdataSourceTriggerEvent
        : public IValidity, IDefault, IDisposable, ICloneable
```

Remarks:

The purpose of this interface is to allow methods that deal with trigger event properties common to all kinds of data source trigger events.

3.18.1 IDataSourceTriggerEvent Properties

3.18.1.1 IDataSourceTriggerEvent.Definition

Description:

The expression string representing the event's definition.

Declaration Syntax:

Visual Basic

```
Property Definition As String
```

C#

```
String Definition { set; get; }
```

C++

```
void set_Definition ( String* expression )
String* get_Definition ()
```

Arguments:

`expression` – string that defines a trigger event.

Exceptions Thrown:

TlaTriggerResourceException:
Condition: Invalid expression string.
Message: The expression does not represent a valid trigger event definition.

TlaTriggerResourceException:
Condition: The expression causes event usage to exceed what trigger definition can support.
Message: The event cannot be used because it requires resources currently unavailable.

Remarks:

Setting the property throws an exception whenever IDataSourceTriggerEvent.IsExpressionValid () returns false for the specified expression string.

Examples:

```
See examples for ILATriggerEvent.
```

3.18.2 IDataSourceTriggerEvent Methods

3.18.2.1 IDataSourceTriggerEvent.IsExpressionValid

Description:

Indicates whether the specified string is a valid event expression.

Declaration Syntax:

Visual Basic

```
Function IsExpressionValid ( expression As String ) As Boolean
```

C#

```
bool IsExpressionValid ( String expression )
```

C++

```
bool IsExpressionValid ( String* expression )
```

Arguments:

`expression` – string expression that defines a trigger event.

Return Value:

Value of true indicates that the expression is a valid event definition.

Exceptions Thrown:

None.

Remarks:

This function parses the string argument to determine whether it is syntactically correct. It also determines whether the tokens in the string refer to elements present in the data source's definition. For instance, an expression that defines a group event for a TLA logic analyzer module must refer to a group that is defined for the data source.

The function does not determine whether the trigger definition can currently support the request for trigger event resources specified in the expression.

Examples:

```
See examples for ILATriggerEvent.
```

3.18.3 IDataSourceTriggerEvent Events

3.18.3.1 IDataSourceTriggerEvent.DefinitionChanged

Description:

When a trigger event’s string representation changes, the DefinitionChanged event is raised.

Declaration Syntax:

Visual Basic

```
Event DefinitionChanged As ObjectEventHandler
```

C#

```
event ObjectEventHandler DefinitionChanged;
```

C++

```
__event  ObjectEventHandler DefinitionChanged;
```

Event Data:

The IDataSourceTriggerEvent object is passed as a parameter.

Remarks:

In addition to a caller’s explicit alteration of the IDataSourceTriggerEvent.Definition property, changes elsewhere in a data source’s definition can modify a trigger event’s expression. Any such change will raise the DefinitionChanged event, regardless of the cause.

Note that changes in a data source’s setup may alter a trigger event’s acquisition behavior without altering the event’s definition. Such changes will not raise an event.

Examples:

```
See examples for ILATriggerEvent.
```

3.19 IDataSourceTriggerAction

Description:

This is the abstract (pure virtual) base interface for all data source trigger actions, including those for Logic analyzer, DSO, and plug-in trigger definitions. IDataSourceTriggerAction contains only those members that must be implemented by all data sources.

Namespace: Tektronix.LogicAnalyzer.Common

Declaration Syntax:

Visual Basic

```
Interface IDataSourceTriggerAction
        Inherits IValidity
        Inherits IDefault
        Inherits IDisposable
        Inherits ICloneable
```

C#

```
interface IdataSourceTriggerAction
        : IValidity, IDefault, IDisposable, ICloneable
```

C++

```
__gc __interface IdataSourceTriggerAction
        : public IValidity, IDefault, IDisposable, ICloneable
```

Remarks:

The purpose of this interface is to allow methods that deal with trigger action properties common to all kinds of data source trigger actions.

3.19.1 IDataSourceTriggerAction Properties

3.19.1.1 IDataSourceTriggerAction.Definition

Description:

The expression string representing the action's definition.

Declaration Syntax:

Visual Basic

```
Property Definition As String
```

C#

```
String Definition { set; get; }
```

C++

```
void set_Definition ( String* expression )
String* get_Definition ()
```

Arguments:

`expression` – string that defines a trigger action.

Exceptions Thrown:

ArgumentException :
Condition: Invalid expression string.
Message: The expression does not represent a valid trigger action definition.

TlaTriggerResourceException:
Condition: The expression causes action usage to exceed what trigger definition can support.
Message: The action cannot be used because it requires resources currently unavailable.

Remarks:

Setting the property throws an exception whenever IDataSourceTriggerAction.IsExpressionValid() returns false for the specified expression string.

Examples:

```
See examples for ILATriggerAction.
```

3.19.2 IDataSourceTriggerAction Methods

3.19.2.1 IDataSourceTriggerAction.IsExpressionValid

Description:

Indicates whether the specified string is a valid action expression.

Declaration Syntax:

Visual Basic

```
Function IsExpressionValid ( expression As String ) As Boolean
```

C#

```
bool IsExpressionValid ( String expression )
```

C++

```
bool IsExpressionValid ( String* expression )
```

Arguments:

`expression` – string expression that defines a trigger action.

Return Value:

Value of true indicates that the expression is a valid action definition.

Exceptions Thrown:

None.

Remarks:

This function parses the string argument to determine whether it is syntactically correct. It also determines whether the tokens in the string refer to elements present in the data source's definition. For instance, an expression that specifies an action involving a system signal must refer to a signal that is defined for the system to which the data source is connected.

The function does not determine whether the trigger definition can currently support the request for trigger action resources specified in the expression.

Examples:

```
See examples for ILATriggerAction.
```

3.19.3 IDataSourceTriggerAction Events

3.19.3.1 IDataSourceTriggerAction.DefinitionChanged

Description:

When a trigger action's string representation changes, the DefinitionChanged event is raised.

Declaration Syntax:

Visual Basic

```
Event DefinitionChanged As ObjectEventHandler
```

C#

```
event ObjectEventHandler DefinitionChanged;
```

C++

```
__event  ObjectEventHandler DefinitionChanged;
```

Event Data:

The IDataSourceTriggerAction object is passed as a parameter.

Remarks:

In addition to a caller's explicit alteration of the IDataSourceTriggerAction.Definition property, changes elsewhere in a data source's definition can modify a trigger action's expression. Any such change will raise the DefinitionChanged event, regardless of the cause.

Note that changes in a data source's definition may alter a trigger action's behavior without altering the action's definition. Such changes will not raise an event.

Examples:

```
See examples for ILATriggerAction.
```

4 Plug-In Interfaces

This section contains .NET interfaces that are implemented only by plug-in classes.

4.1 IPlugIn

Description:

This is the base interface that specifies members common to all types of plug-ins that are designed to work within the TLA application. After instantiating a plug-in, the TLA uses this interface to initialize the plug-in and discover it's basic properties.

Namespace: Tektronix.LogicAnalyzer.PlugIns

Declaration Syntax:

Visual Basic

```
Interface IPlugIn
      Inherits IValidity
      Inherits IDisposable
```

C#

```
interface IPlugIn : IValidity, IDisposable
```

C++

```
__gc __interface IPlugIn : IValidity, IDisposable
```

Applicable Attributes for Plug-In Developers:

Note: Attributes that are applicable to IPlugIn class implementations are also applicable to classes that implement interfaces derived from IPlugIn.

PlugInIdentityAttribute – This attribute allows the a plug in to specify how it is identified to users. The attribute determines the name of the plug-in as it should appear in the TLA's user interface, an icon that should appear next to the name, a group to which the plug-in belongs, and an icon for the group.

PlugInInstantiationAttribute – When applied to a class implementation, this attribute determines if the TLA application will attempt to create more than one instance of a plug-in.

PlugInHelpFileAttribute – If a plug-in has a custom help manual, this attribute describes the name and location of the help manual.

Remarks:

IPlugIn is the minimum interface that a plug-in class must implement. Plug-ins that directly implement this interface are generally called "Tools." When a Tool plug-in class is decorated with PlugInIdentityAttribute, the TLA will place a menu item with this name into the Tools menu and the toolbar. The exact location where users access the menu item and toolbar button can be specified by also setting the "GroupName" property of PlugInNameAttribute.

All plug-ins that need to be instantiated from the TLA user interface must decorate their implementation class with PlugInIdentityAttribute. Without this attribute the TLA application cannot determine how to identify the plug-in to users who might want to instantiate it.

Constructors cannot be specified by the interface definition. The TLA instantiates all plug-ins by creating them with the default constructor. Therefore all class implementations of IPlugIn (or any of its derived interfaces) must have a default constructor.

Although some plug-ins might not use unmanaged resources, all plug-ins must implement the IDisposable interface and conform to its expected behavior. Whenever a plug-in instance is removed from the current system, the TLA calls its Dispose method.

4.1.1 IPlugIn Properties

4.1.1.1 IPlugIn.IsValid

Description:

Gets a Boolean indication whether or not the plug-in is in a valid state. A plug-in should always set this property to false after it has been disposed.

Declaration Syntax:

Visual Basic

```
ReadOnly Property IsDisposed As Boolean
```

C#

```
bool IsDisposed { get; }
```

C++

```
bool get_IsDisposed ();
```

Arguments:

None.

Exceptions Thrown:

None

Remarks:

When a plug-in indicates it is invalid, no client should attempt to use the plug-in for any purpose.

4.1.1.2 IPlugIn.IsGarbage

Description:

The value of this Boolean property indicates whether or not the plug-in has been disposed and is permanently invalid. When IsGarbage is true, IsValid must be false.

Declaration Syntax:

Visual Basic

```
ReadOnly Property IsGarbage As Boolean
```

C#

```
bool IsGarbage { get; }
```

C++

```
bool get_IsGarbage ();
```

Arguments:

None.

Exceptions Thrown:

None.

Remarks:

When a plug-in indicates it is garbage, no client should attempt to use the plug-in for any purpose. Instead all clients of an invalid plug-in should remove their references to it.

4.1.1.3 IPlugIn.PrimaryForm

Description:

This is a reference to the primary Windows Form of the plug-in’s user interface. It may be null, which indicates that the plug-in has no active user interface.

Declaration Syntax:

Visual Basic

```
ReadOnly Property IsPersistent As Boolean
```

C#

```
bool IsPersistent { get; }
```

C++

```
bool get_IsPersistent ();
```

Arguments:

None.

Exceptions Thrown:

None.

Remarks:

This property is used by the TLA application in several ways to maintain it’s own user interface:

If both IPlugIn.PrimaryForm and IPlugIn.PlugInMenu are not null, the application will subscribe to the Form.Activated event. Whenever the primary form of a plug-in is activated, the application will add it’s PlugInMenu to the main application menu (if IPlugIn.PlugInMenu is not null).

Repeatedly pressing the F10 button in the application normally cycles through all the Setup windows. If the PrimaryForm is not null, the form will participate in F10 Setup window cycling.

4.1.1.4 IPlugIn.PlugInMenu

Description:

Gets the menu associated with the plug-in. The TLA will add this menu item to its main menu whenever the plug-in’s primary form has the focus. This property may be null.

Declaration Syntax:

Visual Basic

```
ReadOnly Property PlugInMenu As MenuItem
```

C#

```
MenuItem PlugInMenu { get; }
```

C++

```
MenuItem* get_PlugInMenu  ();
```

Arguments:

None.

Exceptions Thrown:

This property never throws an exception.

Remarks:

This property is allowed to be null. Not all plug-ins require items in the main application menu. For example, an IPlugIn implementation can have a form with its own menu, in which case an item in the main application menu isn’t necessary.

4.1.2 IPlugIn Methods

4.1.2.1 IPlugIn.Initialize

Description:

The TLA application calls this method to establish two-way connection between the TLA application and a new instance of a plug-in.

Declaration Syntax:

Visual Basic

```
Function Initialize (support As ITlaPlugInSupport, _
      userInit As Boolean ) _
      As Boolean
```

C#

```
bool Initialize (ITlaPlugInSupport support, bool userInit)
```

C++

```
bool Initialize (ITlaPlugInSupport* support, bool userInit)
```

Arguments:

`support` – An ITlaPlugInSupport object that provides full access to TPI.NET and to other TLA services that support plug-ins.

`userInit` – A Boolean indicating whether the request to create the plug-in was initiated by a user. The value is true if plug-in initialization was initiated through the TLA UI. The value is false if the plug-in is being created by a programmatic request.

Return Value:

A Boolean that indicates whether or not the plug-in instance was successfully initialized. One reason for failure could be that the component requires modules or other resources that are not available in the current system.

Exceptions Thrown:

This method never throws an exception.

Remarks:

The TLA instantiates plug-in classes using their default constructors. Since the constructor cannot take arguments, the Initialize method is used to pass plug-ins a reference to the TLA's plug-in support interface.

If initialization fails, the plug-in is responsible for indicating failure to the user (typically with a message box). The TLA will not notify the provide its own failure messages, but it will dispose of the component and remove all it's references to the component instance.

The TLA application will call this method only once for each plug-in instance it creates.

4.1.2.2 IPlugIn.Dispose

Description:

Frees all unmanaged resources used by a plug-in instance. The TLA calls this method when removing a plug-in instance from the system.

Declaration Syntax:

Visual Basic

```
Sub Dispose ()
```

C#

```
void Dispose ()
```

C++

```
void Dispose ()
```

Arguments:

None.

Return Value:

None.

Exceptions Thrown:

None.

Remarks:

The IPlugIn specification requires all plug-ins to implement IDisposable, so all plug-ins must implement this method. When the plug-in is disposed, it must set IsValid to false and IsGarbage to true. Then it must raise its ValidityChanged event.

4.1.3 IPlugIn Events

4.1.3.1 IPlugIn.ValidityChanged

Description:

A plug-in raises this event whenever its IsValid or IsGarbage properties change. Plug-ins are expected to become invalid only once during their lifetime, at the time they are disposed. Plug-in implementations use this event to notify that they have be been disposed..

Declaration Syntax:

Visual Basic

```
Event Disposed As SimpleEventHandler
```

C#

```
event SimpleEventHandler Disposed;
```

C++

```
__event SimpleEventHandler Disposed;
```

Event Data:

This passes no parameters.

Remarks:

Any object that maintains a long-term reference to a plug-in should subscribe to this event. After the event is fired, the plug-in is not valid. In general, subscribing objects should handle this event by removing all references to the object that raised the event.

4.2 IDataWindowPlugIn

Description:

This interface is implemented by plug-ins that display data to users. Like listing or waveform windows, a data window plug-in can have a graphical control in the system window. Also like listing or waveform windows, some data window plug-ins can be locked to other data windows.

Namespace: Tektronix.LogicAnalyzer.PlugIns

Declaration Syntax:

Visual Basic

```
Interface IDataWindowPlugIn
      Inherits IPlugIn
      Inherits IDataWindow
```

C#

```
interface IDataWindowPlugIn: IPlugIn, IDataWindow
```

C++

```
__gc __interface IDataWindowPlugIn: public IPlugIn, IDataWindow
```

Applicable Attributes for Plug-In Developers:

All attributes that are applicable to IPlugIn are applicable to IDataWindowPlugIn.

Remarks:

Much of the IDataWindowPlugIn interface is inherited from its base interfaces IPlugIn and IDataWindow. Data window plug-in objects can be locked to other windows by also deriving from ILockingWindow, but they are not required to do so. A class implementation should only derive from ILockingWindow if it can meaningfully implement that interface.

4.2.1 IDataWindowPlugIn Properties

4.2.1.1 IDataWindowPlugIn.SystemWindowControl

Description:

Gets a reference to the System.Windows.Forms.UserControl object that is to represent the data window plug-in in the system window. This property is allowed to be null, which tells the application not to place any information about the plug-in in the system window.

Declaration Syntax:

Visual Basic

```
ReadOnly Property SystemWindowControl As UserControl
```

C#

```
UserControl SystemWindowControl { get; }
```

C++

```
UserControl* get_SystemWindowControl ();
```

Arguments:

None.

Exceptions Thrown:

None.

Remarks:

The SystemWindowControl is a custom control that is placed in the system window to represent the data window component to users. Data window plug-ins are not required to implement a control for system window, so this property may be null. Classes that do provide an are constrained to make it fit in the same amount of space as that given to the system window controls for built-in data windows like Listing and Waveform. If a control is too large, it will be clipped with no scroll bar provided.

4.3 IDataSourcePlugIn

Description:

This interface is implemented by classes that can provide data to clients or can participate in system data acquisition. Like other data sources, a data source plug-in may have a graphical control in the system window.

Namespace: Tektronix.LogicAnalyzer.PlugIns

Declaration Syntax:

Visual Basic

```
Interface IDataSourcePlugIn
      Inherits IPlugIn
      Inherits IDataSource
```

C#

```
interface IDataSourcePlugIn : IPlugIn, IDataSource
```

C++

```
__gc __interface IDataSourcePlugIn : public IPlugIn, public IDataSource
```

Applicable Attributes for Plug-In Developers:

All attributes applicable to IPlugIn are also applicable to IDataSourcePlugIn

Remarks:

Much of the IDataSourcePlugIn interface is defined by its base IPlugIn and IDataSource interfaces.

4.3.1 IDataSourcePlugIn Properties

4.3.1.1 IDataSourcePlugIn.SystemWindowControl

Description:

Gets a reference to the System.Windows.Forms.UserControl object that is to represent the data source plug-in in the system window. This property is allowed to be null, which tells the application not to place any information about the plug-in in the system window.

Declaration Syntax:

Visual Basic

```
ReadOnly Property SystemWindowControl As UserControl
```

C#

```
UserControl SystemWindowControl { get; }
```

C++

```
UserControl* get_SystemWindowControl ();
```

Arguments:

None.

Exceptions Thrown:

None.

Remarks:

The SystemWindowControl is a custom control that is placed in the system window to represent the data source component to users. Data window plug-ins are not required to implement a control for system window, so this property may be null. Classes that do provide an are constrained to make it fit in the same amount of space as that given to the controls representing built-in data sources like LAs and DSOs. If a control is too large, it will be clipped with no scroll bar provided.

4.4 IPlugInSizeEstimate

Description:

Plug-ins that implement ISerializable can optionally implement this interface. IPlugInSizeEstimate provides a means for serializable plug-ins to estimate their contribution to the size of a system file.

Before the TLA saves a system, an estimate of the total system file size is calculated. This estimate is often presented to users through the TLA user interface, and it is also used to determine if there is sufficient disk space to save the system. To create an accurate estimate of the system file size, the TLA needs an estimate of the size of saved plug-in data that will be serialized to storage. Plug-ins provide this value by implementing IPlugInSizeEstimate and its member function GetSerialzedSize(). When a system save is about to occur, the TLA will sum the size estimates of all serializable plug-ins that also implement IPlugInSizeEstimate. This sum will be added to size of TLA data that will saved to produce the full system size estimate.

Namespace: Tektronix.LogicAnalyzer.PlugIn

Declaration Syntax:

Visual Basic

```
Interface IPlugInSizeEstimate
```

C#

```
interface IPlugInSizeEstimate
```

C++

```
__gc __interface IPlugInSizeEstimate
```

Remarks:

Although serializable plug-ins are not required to implement this interface, doing so can greatly improve the accuracy of the system file size estimate that is presented to users in the TLA user interface. Plug-in developers are encouraged to implement IPlugInSizeEstimate whenever they implement ISerializable.

4.4.1 IPlugInSizeEstimate Methods

4.4.1.1IPlugInSizeEstimate.GetSerializedSize

Description:

This method returns the estimated number of bytes required store the serialized plug-in to disk. When a system is about to be saved, serializable plug-ins use this method to tell the TLA application how many bytes the plug-in will contribute to the saved system.

Data source plug-ins should not include the size of data that will be saved as part of the ISaveData interface. Users can choose not to save data from data sources, so including the size of data source data will often overestimate the saved plug-in size. Data sources that Implement ISaveData should use ISaveData.GetDataSize().

Declaration Syntax:

Visual Basic

```
Function GetSerializedSize () As Long
```

C#

```
long GetSerializedSize ();
```

C++

```
__int64 GetSerializedSize ();
```

Arguments:

None.

Return Value:

The estimated number of bytes that will be serialized into the stream when ISerializable.GetObjectData() is called on the plug-in.

Exceptions Thrown:

This method should not throw exceptions.

Remarks:

The return value of this function will always be an estimate. The SerializationInfo object that is provided to a plug-in through its GetObjectData() implementatioc will insert a number of extra bytes into the stream each time the plug-in calls SerializationInfo.AddValue(). Plug-ins should attempt to account for this overhead, but return value of GetSerializedSize() does not need to be perfectly accurate to be useful.

4.5 ISaveData

Description:

This interface supports transfer of raw acquisition data bytes to and from the TLA save file.

Data source plug-ins that are savable can optionally implement this Interface in order to save their acquisition data. When a system is saved with acquisition data, the TLA application uses this interface to provide a stream into which the plug-in can save its data. No assumptions are made by the TLA application about the nature of the data.

When a saved data source plug-in is deserialized, the SavedDataStream property of this interface provides a Stream that contains the acquisition data that the plug-in saved.

Namespace: Tektronix.LogicAnalyzer.TpiNet

Declaration Syntax:

Visual Basic

```
Interface ISaveData
```

C#

```
interface ISaveData
```

C++

```
__gc __interface ISaveData
```

Remarks:

The TLA and the data source plug-in work together with ISaveData interface to save data. First the TLA checks the value of the CanSaveData property. If it is 'true' the TLA calls ISaveData.PrepareToSaveData(). This indicates that the TLA is about to start the process of saving all of the plug-in acquisition data. The plug-in should reset its internal state so that when ISaveData.SaveData() is called, it will start saving at the beginning of its acquisition data.

Next the TLA will call SaveData() and pass a Stream object to the plug-in. The ISaveData implementation then writes a reasonable amount of its data into the stream. The amount of data written should take less than a second to write to a hard disk . If the plug-in has more data to save, the TLA will call SaveData() again after it has processed any UI commands that have occurred while the plug-in was writing to the stream. SaveData() is repeated called until the plug-in indicates that all the data has been written to the stream.

4.5.1 ISaveData Properties

4.5.1.1 ISaveData.CanSaveData

Description:

This is a Boolean property that indicates whether the plug-in is capable of saving meaningful acquisition data through its ISaveData interface. If the plug-in does not currently have any data, or if that data is not in a valid invalid state, the value of this property should be 'false'. If valid acquisition data is available to be saved, this property should be 'true'.

This value affects whether or not the TLA will attempt to save acquisition data from this plug-in during a system save. If the value of CanSaveData is false at the time of a system save, only the plug-in state will be saved through its ISerializable interface implementation.

Declaration Syntax:

Visual Basic

```
ReadOnly Property CanSaveData As Boolean
```

C#

```
bool CanSaveData { get; }
```

C++

```
bool get_CanSaveData();
```

Arguments:

None

Exceptions Thrown:

This property does not throw exceptions.

4.5.1.2 ISaveData.SavedDataStream

Description:

The value of the this property is a Stream object from which a deserialized data source plug-in can access the acquisition data that it contained when it was saved. When a serializable data source plug-in is deserialized, the TLA application sets this property before the Initialize() method of the plug-in is called.

The plug-in may use the Stream object during the full course of its lifetime.

If a data source plug-in has not been created through deserialization, or if no acquisition data was saved, this value will be set to null before Initialize() is called.

Declaration Syntax:

Visual Basic

```
Property SavedDataStream As Stream
```

C#

```
Stream SavedDataStream { set; get; }
```

C++

```
Stream* get_SavedDataStream ();
void set_SavedDataStream (Stream*);
```

Arguments:

None.

Exceptions Thrown:

This property does not throw exceptions.

4.5.2 ISave Data Methods

4.5.2.1ISaveData.PrepareToSaveData

Description:

The TLA calls an implementation of this method to indicate that it is about start saving the data source plug-in acquisition data. After this method is called, the TLA will call the ISaveData.SaveData() to stream the first set of acquisition bytes into the saved system file.

Declaration Syntax:

Visual Basic

```
Sub PrepareToSaveData ()
```

C#

```
void  PrepareToSaveData();
```

C++

```
void  PrepareToSaveData();
```

Arguments:

None.

Return Value:

None.

Exceptions Thrown:

This method does not throw exceptions.

4.5.2.2 ISaveData.GetDataSize

Description:

This method returns the total number of acquisition data bytes contained within a data source plug-in.

Declaration Syntax:

Visual Basic

```
Function GetDataSize () As Long
```

C#

```
long GetDataSize ();
```

C++

```
__int64 GetDataSize ();
```

Arguments:

None.

Return Value:

The return value of this method should approximate the total size of the saved data that would be contributed to a system file if the plug-in were saved in its current state.

Exceptions Thrown:

This method does not throw exceptions.

Remarks:

The TLA uses this method to approximate the size of a system file when the system is saved with its acquisition data.

4.5.2.3 ISaveData.SaveData

Description:

The TLA application calls this method during a system save to allow a data source plug-ins to save its acquisition data into the system file. A stream object is passed to the plug-in as method parameter, and the plug-in writes its acquisition data into this file.

Because data sources can have large quantities of data, the saved data should not be written to the stream all at once. Doing so will likely hang the TLA user interface. To compensate the TLA uses the SaveData method in a way that facilitates writing data to the stream in separate, relatively small chunks. The TLA application will repeatedly call SaveData() until it returns a ’false’ Boolean value. Between calls, the TLA will be able to process user interface commands.

Declaration Syntax:

Visual Basic

```
Function SaveData (outputStream As Stream) As Boolean
```

C#

```
bool SaveData (Stream outputStream)
```

C++

```
bool SaveData (Stream* outputStream)
```

Arguments:

outputStream – This is the stream into which the plug-in writes its acquisition data.

Return Value:

The plug-in should return 'true' if it needs to save more data. When 'true' is returned, SaveData() will be called again, and the same stream will be passed to the plug-in, allowing it to continue saving data from where it left off.

After a plug-in has saved all its acquisition data, it should return 'false'. When 'false' is returned the TLA will not call SaveData() on this plug-in again unless the system saved again.

Exceptions Thrown:

This method does not throw exceptions.

Remarks:

The plug-in is responsible for returning from this method before a noticeable amount of time has passed. If there is too much data to save in a small amount of time, the plug-in should save a small chunk of the data, note where it left off saving data, and return true. The TLA will continue to call SaveData() until the plug-in returns false, meaning there is no more data to save.

In some cases, the user might abort a save operation. In this case the TLA will not call SaveData() again, even if the plug-in returned 'true'. When the system is saved again, the plug-in will be informed that it should restart its saving process from the beginning when its PrepareToSaveData() method is called.

4.6 IImportExportPlugIn

Description:

This interface supports importing/exporting portions of a LA modules setup to/from the TLA application.

Namespace: Tektronix.LogicAnalyzer.TpiNet

Declaration Syntax:

Visual Basic

```
Interface IImportExportPlugIn
```

C#

```
interface IImportExportPlugIn
```

C++

```
__gc __interface IImportExportPlugIn
```

4.6.1 IImportExportPlugIn Methods

4.6.1.1IImportExportPlugIn.OpenExportDialog

Description:

This method opens the export dialog.

Declaration Syntax:

Visual Basic

```
Sub OpenExportDialog (instName As String, owner As IWin32Winodw)
```

C#

```
void OpenExportDialog ([in]String instName, [in]IWin32Window owner)
```

C++

```
void OpenExportDialog ([in]String instName, [in]IWin32Window owner)
```

Arguments:

instName – This specifies instrument user name.
owner – This specifies owner of the dialog window.

Return Value:

None.

Exceptions Thrown:

Exception-Class-Name :
Condition: Condition under which the exception is thrown.
Message: Message string associated with exception.

4.6.1.2IImportExportPlugIn.OpenImportDialog

Description:

This method opens the import dialog.

Declaration Syntax:

Visual Basic

```
Sub OpenImportDialog (instName As String, owner As IWin32Winodw)
```

C#

```
void OpenImportDialog ([in]String instName, [in]IWin32Window owner)
```

C++

```
void OpenImportDialog ([in]String instName, [in]IWin32Window owner)
```

Arguments:

instName – This specifies instrument user name.
owner – This specifies owner of the dialog window.

Return Value:

None.

Exceptions Thrown:

None

5 Structures

5.1 RangeSymbolOptions

Description:

This value type is used to specify the options used when loading a range symbol file with ITlaSystem.LoadSymbolFile.

Namespace: Tektronix.LogicAnalyzer.TpiNet

Declaration Syntax:

Visual Basic

```
<Serializable>
Public Structure RangeSymbolOptions
```

C#

```
[Serializable]
public struct RangeSymbolOptions
```

C++

```
[Serializable]
public __value struct RangeSymbolOptions
```

Remarks:

When a structure of this type is created with the default constructor, the default values of the properties are as follows:

FileFormat = SymbolFileFormat.AutoFormat
TypeOfSymbols = SymbolType.AllSymbols
Bound1 = "0"
Bound2 = "FFFFFFFF"
CustomOffset = null

5.1.1 RangeSymbolOptions Properties

5.1.1.1 RangeSymbolOptions.FileFormat

Description:

Gets or sets the format of a symbol file to be loaded. The value of this property is a member of the SymbolFileFormat enumeration.

Declaration Syntax:

Visual Basic

```
Property FileFormat As SymbolFileFormat
```

C#

```
SymbolFileFormat FileFormat { set; get; }
```

C++

```
SymbolFileFormat get_FileFormat ();
void set_FileFormat (SymbolFileFormat);
```

Arguments:

None.

Exceptions Thrown:

This property does not throw exceptions.

5.1.1.2 RangeSymbolOptions.TypeOfSymbols

Description:

Gets or sets the kind of a symbol file to be loaded. The value of this property is a member of the SymbolType enumeration.

Declaration Syntax:

Visual Basic

```
Property TypeOfSymbols As SymbolType
```

C#

```
SymbolType TypeOfSymbols { set; get; }
```

C++

```
SymbolType get_TypeOfSymbols ();
void set_TypeOfSymbols (SymbolType);
```

Arguments:

None.

Exceptions Thrown:

This property does not throw exceptions.

5.1.1.3 RangeSymbolOptions.Bound1

Description:

Sets or gets one bound of the range of symbols to be loaded by an ITlaSystem.LoadSymbolFile operation. The bound is specified by a string representing 32 bit hexadecimal values from “0” to “FFFFFFFF”. To fully specify the range, both properties Bound1 and Bound2 need to be set.

Declaration Syntax:

Visual Basic

```
Property Bound1 As String
```

C#

```
string Bound1 { set; get; }
```

C++

```
String* get_Bound1 ();
void set_Bound1 (String*);
```

Arguments:

None.

Exceptions Thrown:

This property does not throw exceptions.

Remarks:

Either property Bound1 or Bound2 can be the low bound. Either or both bounds in the range are allowed to be null or empty, and the hexadecimal value is assumed to be 0x0 when the RangeSymbolOptions structure is used in those cases.

5.1.1.4 RangeSymbolOptions.Bound2

Description:

Sets or gets one bound of the range of symbols to be loaded by an ITlaSystem.LoadSymbolFile operation. The bound is specified by a string representing 32 bit hexadecimal values from “0” to “FFFFFFFF”. To fully specify the range, both properties Bound1 and Bound2 need to be set.

Declaration Syntax:

Visual Basic

```
Property Bound2 As String
```

C#

```
string Bound2 { set; get; }
```

C++

```
String* get_Bound2 ();
void set_Bound2 (String*);
```

Arguments:

None.

Exceptions Thrown:

This property does not throw exceptions.

Remarks:

Either property Bound1 or Bound2 can be the low bound. Either or both bounds in the range are allowed to be null or empty, and the hexadecimal value is assumed to be 0x0 when the RangeSymbolOptions structure is used in those cases.

5.1.1.5 RangeSymbolOptions.CustomOffset

Description:

Sets or gets the value of a custom offset. This is specified as a string and takes the form “+N” or “N” or “-N” where N is the 32 bit value of the offset in hexadecimal and can be from "0" to "FFFFFFFF". + indicates that the offset is to be added and – indicates that the offset is to be subtracted. If a sign is not specified, + is assumed.

Declaration Syntax:

Visual Basic

```
Property CustomOffset As String
```

C#

```
string CustomOffset { set; get; }
```

C++

```
String* get_CustomOffset ();
void set_CustomOffset (String*);
```

Arguments:

Argument – Description of argument.

Exceptions Thrown:

Exception-Class-Name:
Condition: Condition under which the exception is thrown.
Message: Message string associated with exception.

Remarks:

The custom offset is not required to be set, when this object is used with ITlaSystem.LoadSymbolFile. If this property is either null or the string is empty, then a default offset of +0 will be used.

5.1.2 RangeSymbolOptions Methods

5.1.2.1 RangeSymbolOptions Constructor

Description:

The RangeSymbolOptions structure is used to pass range symbol options to the method ITlaSystem.LoadSymbolFile. The structure has two constructors. A default constructor allows the properties of the structure to be set after the object is created. The structure also has a parameterized constructor that allows the properties to be set during creation.

Declaration Syntax:

Visual Basic

```
Sub New ()

Sub new (format As SymbolFileFormat, _
         types As SymbolType , _
         rangeBound1 As String, _
         rangeBound2 As String, _
         symbolOffset As String)
```

C#

```
RangeSymbolOptions ()

RangeSymbolOptions (SymbolFileFormat format,
                    SymbolType types,
                    string rangeBound1,
                    string rangeBound2,
                    string symbolOffset )
```

C++

```
RangeSymbolOptions  ()

RangeSymbolOptions (SymbolFileFormat format,
                    SymbolType types,
                    String* rangeBound1,
                    String* rangeBound2,
                    String* symbolOffset )
```

Arguments:

`format` – Specifies the format of the symbol file being loaded.

`fypes` – Specifies the types of symbols to load.

`rangeBound1, rangeBound2` – These arguments specify the range of symbols that will be loaded. Any strings representing hexadecimal values from “0” to “FFFFFFFF” may be specified. If a string is null or empty, then a default value of 0 is assumed.

`symbolOffset` – The value of a custom offset. This is specified as a string and takes the form “+N” or “N” or “-N” where N is the value of the offset in hexadecimal and can be from 0 to

0xFFFFFFFF. + indicates that the offset is to be added, and – indicates that the offset is to be subtracted. If a sign is not specified, + is assumed. If the string is null or empty, then a default offset of +0 is assumed.

Return Value:

None.

Exceptions Thrown:

The constructors don't throw exceptions. If the parameterized constructor is passed bad values, it will set the structure properties to the same values as the default constructor.

Remarks:

The parameterized constructor has one argument for every property of the structure. When the parameterized constructor is used, the parameter values are used to set the corresponding property values.

If the default constructor is used then the default values of the properties are as follows:

```
FileFormat = SymbolFileFormat.AutoFormat
TypeOfSymbols = SymbolType.AllSymbols
Bound1 = "0"
Bound2 = "FFFFFFFF"
CustomOffset = null
```

5.2 InterprobeConnection

Description:

This structure is used to represent a single physical connection between a TLA7AXX analog probe output channel and an oscilloscope input channel.

Namespace: Tektronix.LogicAnalyzer.TpiNet

Declaration Syntax:

Visual Basic

<Serializable>
```
Public Structure InterprobeConnection
```

C#

```
[Serializable]
public struct InterprobeConnection
```

C++

```
[Serializable]
public __value struct InterprobeConnection
```

Remarks:

The TLA cannot by itself determine inter-probing connections between TLA7AXX logic analyzer modules and oscilloscopes. TPI Clients use InterprobeConnection structures to describe inter-probing connections within the TLA.

The connection is defined by a reference to the physical LA module, a reference to the DSO module, the LA output channel connected, and the oscilloscope input channel connected. The connection is set by passing these values to the SetConnection method, which adjusts the structure accordingly.

Changing the structure does not change the System Inter-probing setup until ITlaSystem.SetInterprobeConnection is called with the InterprobeConnection object as an argument.

5.2.1 InterprobeConnection Properties

5.2.1.1 InterprobeConnection.PhysicalModule

Description:

Gets a reference to the TLA7AXX module on the output side of an inter-probe connection.

Declaration Syntax:

Visual Basic

```
ReadOnly Property PhysicalModule As IPhysicalLAModule
```

C#

```
IPhysicalLAModule PhysicalModule { get; }
```

C++

```
IPhysicalLAModule* get_PhysicalModule ();
```

Arguments:

None.

Exceptions Thrown:

None.

Remarks:

All InterprobeConnection property values are set through a call to InterprobeConnection.DefineConnection, which changes all properties of the structure atomically.

5.2.1.2 InterprobeConnection.LAChannel

Description:

Sets or gets the analog probe output channel that is connected.

Declaration Syntax:

Visual Basic

```
ReadOnly Property LAChannel As Int32
```

C#

```
Int32 LAChannel { get; }
```

C++

```
Int32 get_LAChannel ();
```

Arguments:

None.

Exceptions Thrown:

None

Remarks:

The value of this property is in the range 1 – N, where N is the number of analog probe output channels on the module defined by the LAModule property

All InterprobeConnection property values are set through a call to InterprobeConnection.DefineConnection, which changes all properties of the structure atomically.

Examples:

5.2.1.3 InterprobeConnection.OscilloscopeModule

Description:

Gets a reference to the oscilloscope module on the input side of an inter-probe connection.

Declaration Syntax:

Visual Basic

```
ReadOnly Property OscilloscopeModule As IPhysicalModule
```

C#

```
IPhysicalModule OscilloscopeModule { get; }
```

C++

```
IPhysicalModule* get_OscilloscopeModule ();
```

Arguments:

None

Exceptions Thrown:

This property does not throw exceptions.

Remarks:

All InterprobeConnection property values are set through a call to InterprobeConnection.DefineConnection, which changes all properties of the structure atomically.

Examples:

5.2.1.4 InterprobeConnection.OscilloscopeChannel

Description:

Gets the oscilloscope input channel that is connected.

Declaration Syntax:

Visual Basic

```
ReadOnly Property OscilloscopeChannel As Int32
```

C#

```
Int32 OscilloscopeChannel { get; }
```

C++

```
Int32 get_OscilloscopeChannel ();
```

Arguments:

None.

Exceptions Thrown:

Exception-Class-Name:
    Condition: Condition under which the exception is thrown.
    Message: Message string associated with exception.

Remarks:

The value of this property is in the range 1 – N, where N is the number of input channels on the oscilloscope defined by the OscilloscopeModule property

All InterprobeConnection property values are set through a call to InterprobeConnection.DefineConnection, which changes all properties of the structure atomically.

5.2.2 InterprobeConnection Methods

5.2.2.1 InterprobeConnection Constructor

Description:

This default constructor creates an un-initialized InterprobeConnection object. InterprobeConnection objects are value types.

Declaration Syntax:

Visual Basic

```
Sub New ()
```

C#

```
InterprobeConnection()
```

C++

```
InterprobeConnection()
```

Arguments:

None.

Return Value:

None.

Exceptions Thrown:

None

Remarks:

New instances of InterprobeConnection objects are un-initialized. They do not represent a valid inter-probe connection until their DefineConnection method is called with valid parameters.

5.2.2.2 InterprobeConnection.IsValid

Description:

Gets a Boolean indication whether the current property values of the structure represent an inter-probe connection that can be legally set within the TLA system.

Declaration Syntax:

Visual Basic

```
Function IsValid () As Boolean
```

C#

```
Type-Name Method-Name (Type-Name Argument)
```

C++

```
Type-Name* Method-Name (Type-Name* Argument)
```

Arguments:

None.

Return Value:

A Boolean value is returned, indicating whether the structure represents a valid inter-probing connection. Valid inter-probing connections are ones that could be made using the System Inter-probing dialog box.

Exceptions Thrown:

This method does not throw exceptions.

Remarks:

This method indicates only whether the current LA module, DSO module, and module channels represent an inter-probe connection that could legally be made. The connection has not necessarily been set in the system. To register an inter-probe connection with the TLA system, call ITlaSystem.SetInterprobeConnection.

5.2.2.3 InterprobeConnection.DefineConnection

Description:

Creates a representation of an system inter-probing connection. After this method has been successfully called, the InterprobeConnection object can be used in a call to ITlaSystem.SetInterprobeConnection.

Declaration Syntax:

Visual Basic

```
Sub DefineConnection (physicalLA As IPhysicalLAModule _
                      outputChannel As Int32 _
                      oscilloscope As IOScilloscope _
                      inputChannel As Int32)
```

C#

```
void DefineConnection (IPhysicalLAModule physicalLA
                       Int32 outputChannel
                       IOscilloscope oscilloscope
                       Int32 inputChannel)
```

C++

```
void DefineConnection (IPhysicalLAModule* physicalLA
                       Int32 outputChannel
                       IOscilloscope* oscilloscope
                       Int32 inputChannel)
```

Arguments:

`physicalLA` – A reference to the physical LA module that is the source of the inter-probing signal.

`outputChannel` – The channel number of the analog probe output that is connected to the inter-probe cable. The channel number range is 1 – N, where N is the number of analog output channels on the physical LA.

`oscilloscope` – A reference to the oscilloscope module that is the destination of the analog signal.

`inputChannel` – The channel number of the input channel that is connected to the inter-probe cable. The channel number range is 1 – N, where N is the number of analog input channels on the oscilloscope.

Return Value:

None.

Exceptions Thrown:

ArgumentNullException:
Condition: Either the `physicalLA` or `oscilloscope` arguments are null references.
Message: <parameter-name> argument passed as null, but cannot be null.

ArgumentException:
Condition: Invalid physical LA module.
Message: <Module name> does not support inter-probing.

ArgumentOutOfRangeException:
Condition: Either in`putChannel` or `outputChannel` was not in the range of valid channel numbers for the given module.
Message: Channel number not valid.

Remarks:

The physical LA module connected to the oscilloscope must be of a kind that has analog probe outputs. If it is not, then an exception will be thrown.

5.3 RepetitiveSetup

Description:

This structure is used to represent the setup of repetitive acquisitions. The properties of the structure correspond to the setup values in the Repetitive Properties dialog box in the application user interface.

Namespace: Tektronix.LogicAnalyzer.TpiNet

Declaration Syntax:

Visual Basic

Visual Basic

<Serializable>
```
Public Structure RepetitiveSetup
```

C#

```
[Serializable]
public struct RepetitiveSetup
```

C++

```
[Serializable]
public __value struct RepetitiveSetup
```

5.3.1 RepetitiveSetup Properties

5.3.1.1 RepetitiveSetup.SaveFileOptions

Description:

The value of this property is a member of the RepetitiveSaveOptions enumeration.

Declaration Syntax:

Visual Basic

```
Property SaveFileOptions As RepetitiveSaveOptions
```

C#

```
RepetitiveSaveOptions SaveFileOptions { get; }
```

C++

```
RepetitiveSaveOptions get_SaveFileOptions ();
void set_SaveFileOptions (RepetitiveSaveOptions);
```

Arguments:

None.

Exceptions Thrown:

None.

5.3.1.2 RepetitiveSetup.SavedDataSource

Description:

When RepetitiveSetup.SaveFileOptions is set to AllModuleData or UnsuppressedModuleData, this value of this property represents the data source that should be saved to a .TLA file.

Declaration Syntax:

Visual Basic

```
Property SavedDataSource As IDataSource
```

C#

```
IDataSource SavedDataSource{ set; get; }
```

C++

```
IDataSource* get_SavedDataSource ();
void set_SavedDataSource (IDataSource*);
```

Arguments:

None.

Exceptions Thrown:

None.

5.3.1.3 RepetitiveSetup.ExportedObject

Description:

When RepetitiveSetup.SaveFileOptions is set to ExportData, this value of this property represents the object whose data is to be exported. Typically the object is of type IDataWindow, but the object can be of any type that implements IExportData.

Declaration Syntax:

Visual Basic

```
Property ExportedDataWindow As IExportData
```

C#

```
IExportData ExportedDataWindow { set; get; }
```

C++

```
IExportData* get_ExportedDataWindow  ();
void set_ExportedDataWindow  (IExportData*);
```

Arguments:

None.

Exceptions Thrown:

None.

5.3.1.4 RepetitiveSetup.SavePath

Description:

The value of this property is the full pathname of the file that should be saved at the end of each iteration of a repetitive acquisition. This value is only meaningful if the value of RepetitiveSetup.SaveFileOptions is not DoNotSave.

Declaration Syntax:

Visual Basic

```
Property SavePath As String
```

C#

```
string SavePath { set; get; }
```

C++

```
String* get_SavePath ();
void set_SavePath (String*);
```

Arguments:

None.

Exceptions Thrown:

None.

5.3.1.5 RepetitiveSetup.IsNewFile

Description:

The value of this property is a Boolean indication whether a new file should be saved for each iteration of a repetitive acquisition. A new file will be saved if the value is true. If the value is false, a file with the same name will be saved after each iteration. This value is not meaningful if the value of RepetitiveSetup.SaveFileOptions is DoNotSave.

If this value is true, then the StartingSuffix property will be used to generated the first filename.

Declaration Syntax:

Visual Basic

```
Property IsNewFile As Boolean
```

C#

```
bool IsNewFile { set; get; }
```

C++

```
bool get_IsNewFile ();
void set_IsNewFile (bool);
```

Arguments:

None.

Exceptions Thrown:

None.

Remarks:

If the value of this property is false, the same file will be saved each time. This means the file will be overwritten at the end of each iteration of a repetitive acquisition, and only the data from the last acquisition in a repetitive series will be saved.

5.3.1.6 RepetitiveSetup.StartingSuffix

Description:

The value of this property is an integer that will be the first suffix appended to SaveFileName when files are saved at the end of each iteration of a repetitive acquisition. After each successive iteration the integer value is incremented to create a unique name for each file saved during the repetitive acquisition.

Declaration Syntax:

Visual Basic

```
Property StartingSuffix As Integer
```

C#

```
int StartingSuffix { set; get; }
```

C++

```
int get_StartingSuffix ();
void set_StartingSuffix (int);
```

Arguments:

None.

Exceptions Thrown:

None.

5.3.1.7 RepetitiveSetup.IsInfiniteRun

Description:

This Boolean flag indicates whether or not a repetitive acquisition should repeat indefinitely. If the value is true, then a repetitive acquisition will continue until stopped by a user or by a programmatic command; if the value is false, then the next repetitive acquisition will automatically stop after the number of acquisitions specified by the IterationLimit property.

Declaration Syntax:

Visual Basic

```
Property IsInfiniteRun As Boolean
```

C#

```
bool IsInfiniteRun { set; get; }
```

C++

```
bool get_IsInfiniteRun ();
void set_IsInfiniteRun (bool);
```

Arguments:

None.

Exceptions Thrown:

None.

Remarks:

The IRunControl.IterationLimit property to controls how many iterations are performed by when IsInfiniteRun is false.

5.3.1.8 RepetitiveSetup.IterationLimit

Description:

This value of this property is an integer that limits the number of acquisitions to be done by a repetitive acquisition. The default value is 1. If the IsInfiniteRun property is true, then IterationLimit is ignored.

Declaration Syntax:

Visual Basic

```
Property IterationLimit As Integer
```

C#

```
int IterationLimit { set; get; }
```

C++

```
int get_IterationLimit ();
void set_IterationLimit (int);
```

Arguments:

None.

Exceptions Thrown:

None.

Remarks:

The value of this property is expected to be greater than or equal to 1. If the property is set to a value less than one, it will be automatically corrected to be 1.

5.3.1.9 RepetitiveSetup.IsFileOpened

Description:

This Boolean value indicates whether the TLA should open a specific file or program after a repetitive acquisition has completed.

Declaration Syntax:

Visual Basic

```
Property IsFileOpened As Boolean
```

C#

```
bool IsFileOpened { set; get; }
```

C++

```
bool get_IsFileOpened ();
void set_IsFileOpened (bool);
```

Arguments:

None.

Exceptions Thrown:

None.

Remarks:

When the value of this property is true, the following RepetitiveSetup properties are used to determine what file to open and how to open it: OpenFileCommandLine, OpenFileInitialDirectory, and IsOpenFileMinimized.

5.3.1.10 RepetitiveSetup.OpenFileCommandLine

Description:

This property is a string that represents the file to be opened when the OpenFileWhenStopped property is set to true. The file can be a program or a document. The string should contain the full path to the file. If the file is an executable, then the string should also contain any desired command line arguments. The application will the use this command line to open the file after repetitive acquisitions complete.

Declaration Syntax:

Visual Basic

```
Property OpenFileCommandLine As String
```

C#

```
string OpenFileCommandLine { set; get; }
```

C++

```
String* get_OpenFileCommandLine ();
void set_OpenFileCommandLine (String*);
```

Arguments:

None.

Exceptions Thrown:

None.

5.3.1.11 RepetitiveSetup.OpenFileInitialDirectory

Description:

This property is a string representing the path of the initial directory that should be used by the application that is opened after a repetitive acquisition completes. This property is used by the TLA application only when the OpenFileWhenStopped property is true and the OpenFileCommandLine property is not null.

Declaration Syntax:

Visual Basic

```
Property OpenFileInitialDirectory As String
```

C#

```
Type-Name OpenFileInitialDirectory { set; get; }
```

C++

```
Type-Name* get_OpenFileInitialDirectory ();
void set_OpenFileInitialDirectory (Type_Name*);
```

Arguments:

None.

Exceptions Thrown:

None.

5.3.1.12 RepetitiveSetup.IsOpenFileMinimized

Description:

This Boolean property controls how a file is opened after a repetitive acquisition completes. If IsOpenFileMinimized is true, then the file will be opened minimized; otherwise it will be opened in a normal window.

If IsFileOpened is false, then this property is ignored.

Declaration Syntax:

Visual Basic

```
Property IsOpenFileMinimized As Type-Name
```

C#

```
Type-Name IsOpenFileMinimized { set; get; }
```

C++

```
Type-Name* get_IsOpenFileMinimized ();
void set_IsOpenFileMinimized (Type_Name*);
```

Arguments:

None.

Exceptions Thrown:

None.

5.3.2 RepetitiveSetup Methods

5.3.2.1 RepetitiveSetup.IsValid

Description:

This method returns a Boolean indication whether or not the settings currently defined in this object are valid.

Declaration Syntax:

Visual Basic

```
Function IsValid () As Boolean
```

C#

```
bool IsValid ()
```

C++

```
bool IsValid ()
```

Arguments:

None.

Return Value:

Returns true if the settings in this object are valid. Returns false if using this object to set ITlaRunControl.RepetitiveProperties would cause an exception.

Exceptions Thrown:

None.

Remarks:

Only valid RepetitiveSetup objects can be used to set ITlaRunControl.RepetitiveProperties. This method can be used to ensure that setting RepetitiveProperties will not cause an exception.

5.3.2.2 RepetitiveSetup.AddCompared

Description:

This method adds a module to the list of modules whose comparisons to reference data can stop a repetitive acquisition. The method specifies the LA module and whether an equal or not equal comparison will stop the acquistion.

Declaration Syntax:

Visual Basic

```
Sub AddCompare (module As ILaModule, _
      operator As RepetitiveCompareOperator)
```

C#

```
void AddCompare (ILAModule module, RepetitiveCompareOperator operator)
```

C++

```
void AddCompare (ILAModule* module, RepetitiveCompareOperator operator)
```

Arguments:

`module` – The module that will perform a compare to reference data. The module is also the key for removing the comparison later.

`operator` – A member of the RepetiveCompareOperator enumeration. The value determines whether equal comparisons or not equal comparisons to reference data cause repetitive acquisitions to stop.

Return Value:

None.

Exceptions Thrown:

ArgumentNullException :
Condition: The module reference is null.
Message: Null argument passed to AddCompare.

Remarks:

For a RepetitiveSetup to be valid, every module added to the compare list must have a compare enabled in the module setup.

This method will throw an exception if a null argument is passed in. However, this is the only error checking done by the method. To determine whether a RepetitiveSetup object is valid, call the RepetitiveSetup.IsValid method.

5.3.2.3 RepetitiveSetup.RemoveCompare

Description:

Removes a reference data comparison from the list of comparisons that can stop a repetitive acquisition. The comparison to remove is specified by module.

Declaration Syntax:

Visual Basic

```
Sub RemoveCompare (module As ILAModule)
```

C#

```
void RemoveCompare (ILAModule module)
```

C++

```
void RemoveCompare (ILAModule* module)
```

Arguments:

`module` – The module whose compare definition is to be removed from the RepetitiveSetup list of reference data comparisons. The module is the key to select which compare to remove.

Return Value:

None.

Exceptions Thrown:

ArgumentNullException :
Condition: The argument is null.
Message: Null argument passed to RemoveCompare.

Remarks:

If the given module is not currently in the list modules used for repetitive comparisons, this method does nothing.

5.3.2.4 RepetitiveSetup.RemoveAllCompares

Description:

Removes all data comparisons from the list of added data comparisons. Calling this method has the affect of creating a repetitive acquisition setup that will not use data comparisons to stop the acquisition.

Declaration Syntax:

Visual Basic

```
Sub RemoveAllCompares ()
```

C#

```
void RemoveAllCompares ()
```

C++

```
void RemoveAllCompares ()
```

Arguments:

None.

Return Value:

None.

Exceptions Thrown:

None.

Remarks:

If no data comparisons have been added to the RepetitiveSetup structure, this method does nothing.

5.3.2.5 RepetitiveSetup.GetComparisons

Description:

Returns an array of all the LA modules that will compare their data to reference data during a repetitive acquisition. If no comparisons have been added to the repetitive setup, then this method returns null. If the return value is not null, then the next repetitive acquistion will stop if any of the comparisons are true.

Declaration Syntax:

Visual Basic

```
Function GetComparisons () As Hashtable
```

C#

```
Hashtable GetComparisons ()
```

C++

```
Hashtable* GetComparisons ()
```

Arguments:

None.

Return Value:

A Hashtable object that contains all the data comparisons added to the RepetitiveSetup structure with the AddCompare method. The collection of Keys in Hashtable are all the modules whose comparisons are being used. The value associated with the key is RepetitiveCompaerOperator value that specifies whether comparisons with reference data are checked for equality or inequality.

If no comparisons have been added, then the method returns null.

Exceptions Thrown:

None.

5.4 ComparePoint

Description:

This value type is used to specify a begin or trigger relative point for compare in the ICompareSetup interface.

Namespace: Tektronix.LogicAnalyzer.TpiNet

Declaration Syntax:

Visual Basic

```
<Serializable>
Public Structure ComparePoint
```

C#

```
[Serializable]
public struct ComparePoint
```

C++

```
[Serializable]
public __value struct ComparePoint
```

Remarks:

When a structure of this type is created with the default constructor, the default values of the properties are as follows:

Offset=0
RelativeTo=RelativeToBegin

5.4.1 ComparePoint Properties

5.4.1.1 ComparePoint.Offset

Description:

The Offset property is a number of samples relative to the RelativeTo point (begin or trigger). A negative Offset means that many samples before the RelativeTo. An Offset of zero means at the RelativeTo point. A positive offset means that many samples after the RelativeTo point.

Declaration Syntax:

Visual Basic

```
Property Offset As Long
```

C#

```
long Offset { get; set;}
```

C++

```
__int64 get_Offset ();
void set_Offset ( __int64 offset );
```

Arguments:

offset – desired offset relative to trigger or begin.

Exceptions Thrown:

None.

5.4.1.2 ComparePoint.RelativeTo

Description:

The RelativeTo property specifies if the offset is relative to the begin sample of the module or to the trigger sample of the module.

Declaration Syntax:

Visual Basic

```
Property RelativeTo As CompareRelativeValue
```

C#

```
CompareRelativeValue RelativeTo { get; set;}
```

C++

```
CompareRelativeValue get_RelativeTo ();
void set_RelativeTo (CompareRelativeValue relative );
```

Arguments:

relative – relative to trigger or begin.

Exceptions Thrown:

None.

5.4.2 ComparePoint Methods

5.4.2.1 ComparePoint Constructor

Description:

The compare point structure allows trigger or begin relative sample points to be passed with compare setup definitions. The structure has two constructors. The default constructor allows the properties of the structure to be set after the structure is created. The structure also has a parameterized constructor to allow the properties to be set at creation.

Declaration Syntax:

Visual Basic

```
Sub New()
```

```
Sub New( offset As Long, relativeTo As CompareRelativeValue )
```

C#

```
ComparePoint()
```

```
ComparePoint( long offset, CompareRelativeValue relativeTo )
```

C++

```
ComparePoint()
```

```
ComparePoint( __int64 offset, CompareRelativeValue relativeTo )
```

Arguments:

offset – a number of sample before (negative) or after (positive) the relativeTo point.

RelativeTo – an anchor of either begin or trigger sample.

Return Value:

None.

Exceptions Thrown:

None.

Remarks:

The default constructor initializes the properties as follows:

    Offset=0
    RelativeTo=RelativeToBegin

5.5 TimeRange

Description:

This structure contains a pair of values that represent the endpoints of a time range. The TimeRange structure can be used to represent time intervals or setup/.hold windows.

Namespace: Tektronix.LogicAnalyzer.TpiNet

Declaration Syntax:

Visual Basic

<Serializable>
```
Public Structure TimeRange
```

C#

```
[Serializable]
public struct TimeRange
```

C++

```
[Serializable]
public __value struct TimeRange
```

5.5.1 TimeRange Properties

5.5.1.1TimeRange.Time1, TimeRange.Time2

Description:

These properties represent the endpoints of the time range.

Declaration Syntax:

Visual Basic

```
Property Time1 As Decimal
Property Time2 As Decimal
```

C#

```
Decimal Time1 { set; get; }
Decimal Time2 { set; get; }
```

C++

```
Decimal get_Time1 ();
void set_Time1 (Decimal);
Decimal get_Time2 ();
void set_Time2 (Decimal);
```

Arguments:

None.

Exceptions Thrown:

None.

Remarks:

Time1 and Time2 are arbitrary time endpoints that are represented by Decimal values. No relational ordering is enforced, meaning Time1 is not necessarily a lower bound.

5.5.2 TimeRange Methods

5.5.2.1TimeRange Constructors

Description:

TimeRange structures have two constructors. One is a default constructor, and the other initializes the values of Time1 and Time2.

Declaration Syntax:

Visual Basic

```
Sub new ()
Sub new (time1 As Decimal, time2 as Decimal)
```

C#

```
TimeRange ()
TimeRange (Decimal time1, Decimal time2)
```

C++

```
TimeRange ()
TimeRange (Decimal time1, Decimal time2)
```

Arguments:

`time1` – Initializes the Time1 property.

`time2` – Initializes the Time2 property.

Return Value:

None.

Exceptions Thrown:

None.

5.6 VoltsRange

Description:

This structure contains a pair of values that represent the endpoints of a voltage range. The VoltsRange structure can be used to represent voltage intervals.

Namespace: Tektronix.LogicAnalyzer.TpiNet

Declaration Syntax:

Visual Basic

```
<Serializable>
Public Structure VoltsRange
```

C#

```
[Serializable]
public struct VoltsRange
```

C++

```
[Serializable]
public __value struct VoltsRange
```

5.6.1 VoltsRange Properties

5.6.1.1VoltsRange.Volts1, VoltsRange.Volts2

Description:

These properties represent the endpoints of the voltage range.

Declaration Syntax:

Visual Basic

```
Property Volts1 As Decimal
Property Volts2 As Decimal
```

C#

```
Decimal Volts1 { set; get; }
Deciman Volts2 ( set; get; )
```

C++

```
Decimal get_Volts1 ();
void set_Volts1 (Decimal);
Decimal get_Volts2 ();
void set_Volts2 (Decimal);
```

Arguments:

None.

Exceptions Thrown:

None.

Remarks:

Volts1 and Volts2 are arbitrary voltage endpoints that are represented by Decimal values. No relational ordering is enforced, meaning Volts1 is not necessarily a lower bound.

5.6.2 VoltsRange Methods

5.6.2.1VoltsRange Constructors

Description:

VoltsRange structures have two constructors. One is a default constructor, and the other initializes the values of Volts1 and Volts2.

Declaration Syntax:

Visual Basic

```
Sub new ()
Sub new (volts1 As Decimal, volts2 as Decimal)
```

C#

```
TimeRange ()
TimeRange (Decimal volts1, Decimal volts2)
```

C++

```
TimeRange ()
TimeRange (Decimal volts1, Decimal volts2)
```

Arguments:

`volts1` – Initializes the Volts1 property.
`volts2` – Initializes the Volts2 property.

Return Value:

None.

Exceptions Thrown:

None.

5.7 CorrelatorPoint

Description:

This structure contains values describing a time correlated data sample. This structure is returned by methods of ICorrelator.

Namespace: Tektronix.LogicAnalyzer.TpiNet

Declaration Syntax:

Visual Basic

```
<Serializable>
Public Structure CorrelatorPoint Inherits IComparable
```

C#

```
[Serializable]
public struct CorrelatorPoint : IComparable
```

C++

```
[Serializable]
public __value struct CorrelatorPoint : public IComparable
```

5.7.1 CorrelatorPoint Properties

5.7.1.1CorrelatorPoint.Item

Description:

This property represents item handle for the correlation item as returned by ICorrelator.AddItem().

Declaration Syntax:

Visual Basic

```
Property Item As Integer
```

C#

```
int Item { set; get; }
```

C++

```
int get_Item ();
void set_Item (int);
```

Arguments:

None.

Exceptions Thrown:

None.

Remarks:

Use IsValid() on ICorrelator return values to determine if properties contain valid values.

5.7.1.2CorrelatorPoint.Rank

Description:

This property represents a weighting to resolve timestamp ties between different correlation items. If samples from two items contain identical timestamps, the item with the smaller Rank is assigned an earlier correlation point than an item with a larger rank. The ICorrelator assigns unique ranks to items.

Declaration Syntax:

Visual Basic

```
Property Rank As Long
```

C#

```
long Rank { set; get; }
```

C++

```
__int64 get_Rank ();
void set_Rank (__int64);
```

Arguments:

None.

Exceptions Thrown:

None.

Remarks:

Use IsValid() on ICorrelator return values to determine if properties contain valid values.

Each CorrelatorPoint in the correlation sequencing contains a unique Rank and Time pair. No other correlation points in the sequencing share the same Rank and Time values.

5.7.1.3CorrelatorPoint.Sample

Description:

This property represents a data source sample number of the Item.

Declaration Syntax:

Visual Basic

```
Property Sample As Long
```

C#

```
long Sample { set; get; }
```

C++

```
__int64 get_Sample ();
void set_Sample (__int64);
```

Arguments:

None.

Exceptions Thrown:

None.

Remarks:

Use IsValid() on ICorrelator return values to determine if properties contain valid values.

5.7.1.4CorrelatorPoint.Time

Description:

This property represents time-aligned timestamps data for the correlation point. The timestamp includes all time adjustments for the time base and can be used for locking window operations.

Declaration Syntax:

Visual Basic

```
Property Time As Decimal
```

C#

```
Decimal Time { set; get; }
```

C++

```
Decimal get_Time ();
void set_Time (Decimal);
```

Arguments:

None.

Exceptions Thrown:

None.

Remarks:

Use IsValid() on ICorrelator return values to determine if properties contain valid values.

5.7.2 CorrelatorPoint Methods

5.7.2.1CorrelatorPoint Constructors

Description:

CorrelatorPoint structures have two constructors. One is a default constructor, and the other initializes the values of the properties. The default constructor calls Invalidate() to initialize properties such that IsValid() returns false.

Declaration Syntax:

Visual Basic

```
Sub new ()
Sub new (item As Integer, rank As Integer, sample As Long, time As
Decimal)
```

C#

```
CorrelatorPoint ()
CorrelatorPoint (int item, int rank, long sample, Decimal time)
```

C++

```
SequenceInfo ()
SequenceInfo (int item, int rank, __int64 sample, Decimal time)
```

Arguments:

`item` – Initializes the Item property.

rank – Initializes the Rank property.

`sample` – Initializes the Sample property.

`time` – Initializes the Time property.

Return Value:

None.

Exceptions Thrown:

None.

5.7.2.2 CorrelatorPoint.IsValid

Description:

Checks to see if the CorrelatorPoint has valid properties. This checks Item, Rank, and Sample properties. If any are negative, the point is considered invalid.

Declaration Syntax:

Visual Basic

```
Function IsValid () As Boolean
```

C#

```
bool IsValid ()
```

C++

```
bool IsValid ()
```

Arguments:

`None`.

Return Value:

False if properties contain invalid values, otherwise true.

Exceptions Thrown:

None.

Remarks:

No check is made against the ICorrelator object to verify that the properties reflect an actual sample.

5.7.2.3 CorrelatorPoint.Invalidate

Description:

Sets the properties to a state representative of an invalid ComparePoint. The Item, Rank, and Sample are set to –1. The Time is set to 0.

Declaration Syntax:

Visual Basic

```
Sub Invalidate ()
```

C#

```
void Invalidate ()
```

C++

```
void Invalidate ()
```

Arguments:

`None`.

Return Value:

None.

Exceptions Thrown:

None.

5.7.2.4 CorrelatorPoint.CompareTo

Description:

This implements IComparable.CompareTo() to allow comparison of two CorrelatorPoint structures.

Declaration Syntax:

Visual Basic

```
Function CompareTo ( obj As Object ) As Integer
```

C#

```
int CompareTo ( object obj )
```

C++

```
int CompareTo ( Object* obj )
```

Arguments:

`obj` – another CorrelatorPoint object to compare against.

Return Value:

Returns less than zero if this instance is less than obj. Returns greater than zero if this instance is greater than obj. Returns zero if this instance is equal to obj.

Exceptions Thrown:

None.

Remarks:

Comparison of CorrelatorPoint structures is based first on Time properties. A CorrelatorPoint with a Time value less than another is considered less than (earlier than) the other. If Time values are equal, the Ranks are compared. A smaller Rank is considered less than (earlier than) the other. If both Time and Rank are identical, the CorrelatorPoint structures are considered equal.

5.8 ChannelImportExportOptions

Description:

This value type is used to specify the options used when importing or exporting channel setup with ILaModule.ImportChannelSetup or ILaModule.ExportChannelSetup.

Namespace: Tektronix.LogicAnalyzer.TpiNet

Declaration Syntax:

Visual Basic

<Serializable>
```
Public Structure ChannelImportExportOptions
```

C#

```
[Serializable]
public struct ChannelImportExportOptions
```

C++

```
[Serializable]
public __value struct ChannelImportExportOptions
```

Remarks:

When a structure of this type is created with the default constructor, the default values of the properties are as follows:

Delimiter = FieldDelimiter.Tab
UserName = true
Polarity = false
Threshold = false
Compare = false
GroupInfo = true
AutoDetect = true

5.8.1 ChannelImportExportOptions Properties

5.8.1.1 ChannelImportExportOptions.Delimiter

Description:

Gets or sets the delimiter of a channel setup file to be imported/exported. The value of this property is a member of the FieldDelimiter enumeration.

Declaration Syntax:

Visual Basic

```
Property Delimiter As FieldDelimiter
```

C#

```
FieldDelimiter Delimiter { set; get; }
```

C++

```
FieldDelimiter get_Delimiter ();
void set_Delimiter (FieldDelimiter);
```

Arguments:

None.

Exceptions Thrown:

This property does not throw exceptions.

5.8.1.2 ChannelImportExportOptions.UserName

Description:

This Boolean property indicates whether user-defined channel name should be imported/exported. True indicates that it is.

Declaration Syntax:

Visual Basic

```
Property UserName As Boolean
```

C#

```
bool UserName { set; get; }
```

C++

```
bool get_UserName ();
void set_UserName (bool);
```

Arguments:

None.

Exceptions Thrown:

This property does not throw exceptions.

5.8.1.3 ChannelImportExportOptions.Polarity

Description:

This Boolean property indicates whether polarity information should be imported/exported. True indicates that it is.

Declaration Syntax:

Visual Basic

```
Property Polarity As Boolean
```

C#

```
bool Polarity { set; get; }
```

C++

```
bool get_Polarity ();
void set_Polarity (bool);
```

Arguments:

None.

Exceptions Thrown:

This property does not throw exceptions.

5.8.1.4 ChannelImportExportOptions.Threshold

Description:

This Boolean property indicates whether threshold information should be imported/exported. True indicates that it is.

Declaration Syntax:

Visual Basic

```
Property Threshold As Boolean
```

C#

```
bool Threshold { set; get; }
```

C++

```
bool get_Threshold ();
void set_Threshold (bool);
```

Arguments:

None.

Exceptions Thrown:

This property does not throw exceptions.

Examples:

5.8.1.5 ChannelImportExportOptions.Compare

Description:

This Boolean property indicates whether compare enable/disable information should be imported/exported. True indicates that it is.

Declaration Syntax:

Visual Basic

```
Property Compare As Boolean
```

C#

```
bool Compare { set; get; }
```

C++

```
bool get_Compare ();
void set_Compare (bool);
```

Arguments:

None.

Exceptions Thrown:

This property does not throw exceptions.

5.8.1.6 ChannelImportExportOptions.GroupInfo

Description:

This Boolean property indicates whether group information should be imported/exported. True indicates that it is.

Declaration Syntax:

Visual Basic

```
Property GroupInfo As Boolean
```

C#

```
bool GroupInfo { set; get; }
```

C++

```
bool get_GroupInfo ();
void set_GroupInfo (bool);
```

Arguments:

None.

Exceptions Thrown:

This property does not throw exceptions.

5.8.1.7 ChannelImportExportOptions.AutoDetect

Description:

This Boolean property indicates whether import parameter should be detected from the imported file. True indicates that it is.

Declaration Syntax:

Visual Basic

```
Property AutoDetect As Boolean
```

C#

```
bool AutoDetect { set; get; }
```

C++

```
bool get_AutoDetect ();
void set_AutoDetect (bool);
```

Arguments:

None.

Exceptions Thrown:

This property does not throw exceptions.

5.8.2 ChannelImportExportOptions Methods

5.8.2.1 ChannelImportExportOptions Constructor

Description:

The ChannelImportExportOptions structure is used to pass channel import/export options to the method ILaModule.ImportChannelSetup and ILaModule.ExportChannelSetup. The structure has two constructors. A default constructor allows the properties of the structure to be set after the object is created. The structure also has a parameterized constructor that allows the properties to be set during creation.

Declaration Syntax:

Visual Basic

```
Sub New ()

Sub new (delimiter As FieldDelimiter, _
         userName As Boolean , _
         polarity As Boolean, _
         threshold As Boolean, _
         compare As Boolean, _
         groupInfo As Boolean, _
         autoDetect As Boolean)
```

C#

```
ChannelImportExportOptions ()

ChannelImportExportOptions (FieldDelimiter delimiter,
                            bool userName,
                            bool polarity,
                            bool threshold,
                            bool compare,
                            bool groupInfo,
                            bool autoDetect)
```

C++

```
ChannelImportExportOptions ()

ChannelImportExportOptions (FieldDelimiter delimiter,
                            bool userName,
                            bool polarity,
                            bool threshold,
                            bool compare,
                            bool groupInfo,
                            bool autoDetect)
```

Arguments:

`delimiter` – Specifies the delimiter for exported file.

`userName` – Specifies if the user-defined channel name should be included.

`polarity` – Specifies if the polarity information should be included.

`threshold` – Specifies if the threshold information should be included.

`compare` – Specifies if the compare enable/disable information should be included.

`groupInfo` – Specifies if the group information should be included.

`autoDetect` – Specifies if the import parameter should be detected from the imported file.

Return Value:

None.

Exceptions Thrown:

The constructors don't throw exceptions. If the parameterized constructor is passed bad values, it will set the structure properties to the same values as the default constructor.

Remarks:

The parameterized constructor has one argument for every property of the structure. When the parameterized constructor is used, the parameter values are used to set the corresponding property values.

If the default constructor is used then the default values of the properties are as follows:

Delimiter = FieldDelimiter.Tab
UserName = true
Polarity = false
Threshold = false
Compare = false
GroupInfo = true
AutoDetect = true

6 Exception Classes

All specialized exception classes that can be thrown by interfaces should be documented. The exception class can be described using the same style as the template for interfaces. See section 1. Exception classes should not have events.

6.1 PlugInException

Description:

Objects that implement IPlugIn throw this exception when exception occurs during the processing of some required properties and methods. This exception distinguishes an exception as one that was thrown by code in a plug-in and not by code in the TLA application.

Namespace: Tektronix.LogicAnalyzer.PlugIns

Declaration Syntax:

Visual Basic

```
Class PlugInException
      Inherits ApplicationException
```

C#

```
class PlugInException : ApplicationException
```

C++

```
__gc class PlugInException : public ApplicationException
```

Remarks:

Developers who want more specialized exceptions to be thrown by a plug-in implementation should derive all exceptions from this class. The TLA depends on plug-ins throwing exceptions of this type.

6.1.1.1 PlugInException.Instance

Description:

Gets a reference to the plug-in instance, that threw the PlugInException.

Declaration Syntax:

Visual Basic

```
Property PlugInInstance As IPlugIn
```

C#

```
IPlugIn PlugInInstance { set; get;}
```

C++

```
IPlugIn* get_PlugInInstance ();
```

Arguments:

None.

Exceptions Thrown:

This property does not throw exceptions.

Remarks:

Plug-in implementations must ensure that the value of this property is not null when they throw a PlugInException.

6.2 TlaPlugInException

Description:

The TLA application throws an exception of this type when a TPI client attempts an action on a plug-in that cannot be completed.

Namespace: Tektronix.LogicAnalyzer.TpiNet

Declaration Syntax:

Visual Basic

```
Class TlaPlugInException
      Inherits ApplicationException
```

C#

```
class TlaPlugInException: ApplicationException
```

C++

```
__gc class TlaPlugInException: public ApplicationException
```

Remarks:

Note that this exception class is thrown by the TLA application and not by a plug-in itself. For instance, TlaPlugInException will be thrown when ITlaSystem.CreatePlugInInstance is called to create a second instance of a single-instance plug-in.

Plug-in implementations should not throw this exception. Plug-in classes must throw exceptions either from the Tektronix.LogicAnalyzers.PlugIns namespace, from the .NET framework, or from exceptions defined in its own assembly.

6.2.1 TlaPlugInException Properties

6.2.1.1 TlaPlugInException.PlugInType

Description:

A reference to the plug-in Type upon which the TLA cannot complete an action.

Declaration Syntax:

Visual Basic

```
Property PlugInType As Type
```

C#

```
Type PlugInType { get; }
```

C++

```
Type* get_PlugInType ();
```

Arguments:

None.

Exceptions Thrown:

This property does not throw exceptions.

Remarks:

The TlaPlugInException can be throw even when an action would improperly affect a whole plug in class, not just instances. This property provides access to type information from the plug-in class that was involved in causing the exception.

This property will not be null even when TlaPlugInException.PlugInInstance is null.

6.2.1.2 TlaPlugInException.Instance

Description:

Gets a reference to the plug-in instance, if any, associated with the TlaPlugInException.

Declaration Syntax:

Visual Basic

```
Property PlugInInstance As IPlugIn
```

C#

```
IPlugIn PlugInInstance { set; get;}
```

C++

```
IPlugIn* get_PlugInInstance ();
```

Arguments:

None.

Exceptions Thrown:

This property does not throw exceptions.

Remarks:

The value of this property is sometimes a null reference. A TlaPlugInException can be thrown without being caused by a specific plug-in instance.

6.3 TlaFileOperationException

Description:

The TLA throws this kind of exception when a TLA-specific kind of failure occurs during a save or load operation. This failure is not a file system problem.

Namespace: Tektronix.LogicAnalyzer.TpiNet.

Declaration Syntax:

Visual Basic

```
Class TlaFileException
      Inherits ApplicationException
```

C#

```
class TlaFileException : ApplicationException
```

C++

```
__gc class TlaFileException : public ApplicationException
```

Remarks:

This exception indicates that the problem is not entirely related to the file system. Instead the problem is related to the state of the TLA at the time of the file operation. For example this exception is thrown when attempting to save the system while the system is running.

The specific kind of failure is indicated by the Failure property. The ToString method also describes the file operation failure.

6.3.1 TlaFileOperationException Properties

6.3.1.1 TlaFileOperationException.Failure

Description:

The value of this property is a member of the FileOperationFailures enumeration. The member indicates the kind of failure that caused the exception.

Declaration Syntax:

Visual Basic

```
Property Property-Name As Type-Name
```

C#

```
Type-Name Property-Name { set; get;}
```

C++

```
Type-Name* get_Property-Name ();
void set_Property-Name (Type_Name*);
```

Arguments:

Argument – Description of argument.

Exceptions Thrown:

Exception-Class-Name:
  Condition: Condition under which the exception is thrown.
  Message: Message string associated with exception.

Remarks:

Specific details about the property.

6.4 TlaNoDataException

Description:

The TLA application throws this exception when no data is available to an operation that requires data.

Namespace: Tektronix.LogicAnalyzer.TpiNet

Declaration Syntax:

Visual Basic

```
Class TlaNoDataException
      Inherits ApplicationException
```

C#

```
class TlaNoDataException : ApplicationException
```

C++

```
__gc class TlaNoDataException : ApplicationException
```

Remarks:

This exception adds no new functionality to the ApplicationException from which it derives. The class itself is enough to distinguish this exception from other exceptions.

6.5 TlaSystemTriggerException

Description:

The TLA throws this exception when it cannot make an expected change to the system trigger.

Namespace: Fully qualified name of namespace that contains the interface.

Declaration Syntax:

Visual Basic

```
Class TlaSystemTriggerException
      Inherits ApplicationException
```

C#

```
class TlaSystemTriggerException : ApplicationException
```

C++

```
__gc class TlaSystemTriggerException : ApplicationException
```

6.6 TlaProhibitedDuringRunException

Description:

The TLA raises an exception of this type when an acquisition is in progress and a TPI client requests a action that cannot be performed during a run.

Namespace: Tektronix.LogicAnalyzer.TpiNet

Declaration Syntax:

Visual Basic

```
Class TlaProhibitedDuringRunException
      Inherits ApplicationException
```

C#

```
class TlaProhibitedDuringRunException : ApplicationException
```

C++

```
__gc __interface TlaProhibitedDuringRunException
      : public ApplicationException
```

Remarks:

This exception inherits directly from System.ApplicationException, and it does not add any properties or methods to its base class. The class name and type are used to distinguish this exception from other ApplicationExceptions.

6.7 TlaTriggerResourceException

Description:

The TLA application throws an exception of this type when a client attempts to use a trigger resource that either is unavailable or is incompatible with the trigger definition's current settings.

Namespace: Tektronix.LogicAnalyzer.Common

Declaration Syntax:

Visual Basic

```
Class TlaTriggerResourceException
      Inherits ApplicationException
```

C#

```
class TlaTriggerResourceException: ApplicationException
```

C++

```
__gc class TlaTriggerResourceException :
                        public ApplicationException
```

Remarks:

Note that this exception class is thrown by IDataSourceTrigger, IDataSourceTriggerState, IDataSourceTriggerClause, IDataSourceTriggerEvent, IdataSourceTriggerAction, or any interface derived from them. For instance, TlaTriggerResourceConflictException will be thrown when a client calls IDataSourceTriggerClause.AddEventBefore any time a call to IDataSourceTriggerClause.CanAddEvent would return false.

6.7.1 TlaTriggerResourceException Properties

6.7.1.1 TlaTriggerResourceException.ResourceType

Description:

A reference to the TriggerResource that caused the exception.

Declaration Syntax:

Visual Basic

```
Property ResourceType As TriggerResource
```

C#

```
TriggerResource ResourceType { get; }
```

C++

```
TriggerResource * get_ResourceType ();
```

Arguments:

None.

Exceptions Thrown:

This property does not throw exceptions.

Remarks:

The TlaTriggerResourceException can be thrown for any type of resource conflict that cannot be resolved by the underlying data source.

This property will never be null.

6.7.1.2 TlaTriggerResourceException.ConflictType

Description:

A reference to the TriggerConflict that caused the exception.

Declaration Syntax:

Visual Basic

```
Property ConflictType As TriggerConflict
```

C#

```
TriggerConflict ConflictType { get; }
```

C++

```
TriggerConflict * get_ConflictType ();
```

Arguments:

None.

Exceptions Thrown:

This property does not throw exceptions.

Remarks:

The TlaTriggerResourceException can be thrown for any type of resource conflict that cannot be resolved by the underlying data source.

This property will never be null.

6.8 TlaUnsupportedSetupException

Description:

The TLA application throws this exception when an attempt is made to alter a setup parameter that is not supported by the instrument type or configuration.

Namespace: Tektronix.LogicAnalyzer.TpiNet

Declaration Syntax:

Visual Basic

```
Class TlaUnsupportedSetupException Inherits ApplicationException
```

C#

```
class TlaUnsupportedSetupException: ApplicationException
```

C++

```
__gc class TlaUnsupportedSetupException: ApplicationException
```

Remarks:

This exception adds no new functionality to the ApplicationException from which it derives. The class itself is enough to distinguish this exception from other exceptions.

7 Enumerations

This section contains all enumerations used by interfaces, classes, and structures defined in this document. Enumerations in .NET are derived from System.Enum.

7.1 SignalsIn

Description:

Enumerates the internal signal types that can be connected to External Signal In.

Namespace: Tektronix.LogicAnalyzer.TpiNet

Declaration Syntax:

Visual Basic

```
Public Enum SignalsIn As Byte
```

C#

```
public enum SignalsIn: Byte
```

C++

```
__value public enum SignalsIn: Byte
```

Enumeration Members:

None – No internal signal connected.

HighSpeedSignal1

HighSpeedSignal2

Signal3

Signal4

Remarks:

The SignalsIn enumeration is used for get and set operations on the property ITlaSystem.ExternalSignalIn.

7.2 SignalsOut

Description:

Enumerates the internal signal types that can be connected to External Signal Out.

Namespace: Tektronix.LogicAnalyzer.TpiNet

Declaration Syntax:

Visual Basic

```
Public Enum SignalsOut As Byte
```

C#

```
public enum SignalsOut: Byte
```

C++

```
__value public enum SignalsOut: Byte
```

Enumeration Members:

None – No internal signal connected.

HighSpeedSignal1

HighSpeedSignal2

Signal3

Signal4

Clock - 10 MHz clock signal

Remarks:

The SignalsOut enumeration is used for get and set operations on the property ITlaSystem.ExternalSignalOut.

7.3 RepetitiveStopReason

Description:

Enumeration of reasons why a repetitive acquisition can stop.

Namespace: Tektronix.LogicAnalyzer.TpiNet

Declaration Syntax:

Visual Basic

```
Public Enum RepetitiveStopReason As Byte
```

C#

```
public enum RepetitiveStopReason : Byte
```

C++

```
__value public enum RepetitiveStopReason : Byte
```

Enumeration Members:

UnknownReason – Repetitive acquisition was never run or stopped for unknown reason.

UserRequest – Repetitive acquisition was halted by a user STOP request, either through the user interface or via TPI.

CountReached – Repetitive acquisition was halted because the repeat count was reached.

CompareSucceeded – Repetitive acquisition was halted because the compare test succeeded.

SaveFailed – Repetitive acquisition was halted because an attempt to save data failed.

Remarks:

A member of this enumeration is returned by the ITlaRunControl.GetRepetitiveStopReason method.

7.4 RepetitiveSaveOptions

Description:

Enumerates the user options for saving a file after the completion of an iteration in a repetitive acquisition.

Namespace: Tektronix.LogicAnalyzer.TpiNet

Declaration Syntax:

Visual Basic

```
Public Enum RepetitiveSaveOptions As Byte
```

C#

```
public enum RepetitiveSaveOptions : byte
```

C++

```
__value public enum RepetitiveSaveOptions : Byte
```

Enumeration Members:

`DoNotSave` – Does not automatically save any files during a repetitive acquisition.

`AllSystemData` – Saves the system with all data.

`UnsuppressedSystemData` – Saves the system, but only saves unsuppressed data.

`AllModuleData` – Saves a specified module with all its data.

`UnsuppressedModuleData` – Saves a specified module, but only saves its unsuppressed data.

`ExportData` – Exports data from a listing window into a file.

Remarks:

The primary use of this enumeration is for setting up a RepetitiveSetup structure, which defines how the TLA performs repetitive acquisitions.

7.5 RepetitiveCompareOperator

Description:

When at least one LA in the system has a compare defined, repetitive acquisitions can be stopped by comparisons of acquired data with reference data. This enumeration specifies the compare operators that are allowed.

Namespace: Tektronix.LogicAnalyzer.TpiNet

Declaration Syntax:

Visual Basic

```
Public Enum RepetitivCompareOperator As Byte
```

C#

```
public enum RepetitivCompareOperator : byte
```

C++

```
__value public enum RepetitivCompareOperator : unsigned char
```

Enumeration Members:

`Equal` – Specifies that a repetitive acquisition should stop if newly acquired data is the same as module-defined reference data.

`NotEqual` – Specifies that a repetitive acquisition should stop if newly acquired data is different than module-defined reference data.

7.6 SaveDataOptions

Description:

Enumerates a user's options for saving data along with TLA saved files. The enumeration members correspond to the Save Options in the Save As dialog.

Namespace: Tektronix.LogicAnalyzer.TpiNet

Declaration Syntax:

Visual Basic

```
Public Enum SaveDataOptions As Byte
```

C#

```
public enum SaveDataOptions : Byte
```

C++

```
__value public enum SaveDataOptions : Byte
```

Enumeration Members:

`AllData` – Save all data from all data sources in the system.

`UnsuppressedData` – Save only unsuppressed data from all data sources in the system.

`NoData` – Save only the system setup, but do not save any data from the system data sources.

Remarks:

This enumeration is as a parameter to TPI.NET methods that save system and module setups.

7.7 FileOperationFailures

Description:

This is an enumeration of TLA-specific reasons why a load or save operation might fail.

Namespace: Fully qualified name of namespace that contains the interface.

Declaration Syntax:

Visual Basic

```
Access-Modifier Enum Enumeration-Name As Type-Name
```

C#

```
Access-Modifier enum Enumeration-Name : Type-Name
```

C++

```
__value Access-Modifier enum Enumeration-Name : Type-Name
```

Enumeration Members:

`SystemRunning` – The system is running while the file operation was attempted.

`OutOfMemory` – There is not enough memory to complete the operation. <NOTE: this might not be needed, but it's here for the moment because it's a failure reason in COM TPI>

`InvalidFile` – The file is invalid. This can occur when reading a corrupted TLA file or if trying to read a file that is not a saved system, module, or trigger setup.

`HardwareMismatch` – The hardware configuration in the file does not match the current hardware configuration. This can happen when loading systems, modules or triggers.

ModuleError – An error occurred while loading a module from the file.

`PlugInError` – An error occurred while loading a plug-in from the file.

`DataError` – An error occurred while loading data from the file.

`DataWindowError –` An error occurred while loading a data window from the file.

`UnknownError` – The file operation could not be completed for an unspecified reason.

Remarks:

A load or save operation can sometimes fail for reasons that are not related to the file system. In those situations, the TLA will generate a TlaFileOperationException. The reason for the exception will be one of the members of this enumeration.

7.8 SymbolFileFormat

Description:

This is an enumeration of symbol file formats supported by the TLA. Symbol files with the enumerated formats can be loaded into the TLA programmatically. Use this enumeration when setting up a RangeSymbolOptions structure for use with the ITlaSystem::LoadSymbolFile Method.

Namespace: Tektronix.LogicAnalyzer.TpiNet

Declaration Syntax:

Visual Basic

<Serializable>
```
Public Enum SymbolFileFormat As Byte
```

C#

[Serializable]
```
public enum SymbolFileFormat: Byte
```

C++

[Serializable]
```
__value public enum SymbolFileFormat: Byte
```

Enumeration Members:

`AutoFormat`

`TSF`

`COFF`

`ELF`

`IEEE695`

`OMF86`

`OMF286`

`OMF386`

`OMF51`

`OMF166`

Remarks:

Specific details about the enumeration.

7.9 SymbolType

Description:

Enumerates the kinds of symbols that can be defined in a symbol file.

Namespace: Tektronix.LogicAnalyzers.TpiNet

Declaration Syntax:

Visual Basic

<Flags>
<Serializable>
```
Public Enum SymbolType As Byte
```

C#

[Flags]
[Serializable]
```
public enum SymbolType : Byte
```

C++

[Flags]
[Serializable]
```
__value public enum SymbolType : Byte
```

Enumeration Members:

`AllSymbols` – Represents all symbol types in symbol files.

`Functions` – Function symbols.

`Variables` – Variable symbols.

`SourceCode` – Source code symbols.

`Colors` – Symbols for coloring.

7.10 SystemTrigger

Description:

Enumerates the pre-defined options for setting up how the system trigger is generated. These options correspond to options in the System Trigger dialog.

Namespace: Tektronix.LogicAnalyzer.TpiNet

Declaration Syntax:

Visual Basic

```
Public Enum SystemTrigger As Byte
```

C#

```
public enum SystemTrigger : Byte
```

C++

```
__value public enum SystemTrigger : Byte
```

Enumeration Members:

`IndependentModules` – All modules trigger themselves, and the system triggers after all modules have triggered.

`AllModules` – All module triggers are set to trigger the system, so the system is triggered by the first module that triggers.

`SingleModule` – The system can be triggered only by one specific module.

Remarks:

7.11 PlugInTriggerMode

Description:

Describes whether an IAcquisition plug-in triggers itself, triggers all modules, or waits for a system trigger during acquisition.

Namespace: Tektronix.LogicAnalyzer.PlugIn

Declaration Syntax:

Visual Basic

```
Public Enum PlugInTriggerMode As Byte
```

C#

```
public enum PlugInTriggerMode : byte
```

C++

```
__value public enum PlugInTriggerMode : Byte
```

Enumeration Members:

`TriggerSelf` – The plug-in will trigger itself during an acquisition.

`TriggerAll` – The plug-in will trigger assert the system trigger when it triggers.

`WaitForSystemTrigger` – The plug-in will trigger neither itself nor the system during an acquisition. It will wait to be triggered by the system trigger.

7.12 LockingWindowItem

Description:

Enumerates the data window elements that are locked affected by locking windows together.

Namespace: Tektronix.LogicAnalyzer.TpiNet

Declaration Syntax:

Visual Basic

```
Public Enum LockingWindowItem As Byte
```

C#

```
public enum LockingWindowItem : Byte
```

C++

```
__value public enum LockingWindowItem : Byte
```

Enumeration Members:

`Cursor1` – Cursor1 in waveform, listing and some data window plug-ins.

`Cursor2` – Cursor2.

`DisplayCenter` – The center of a data window. The exact meaning is defined the type of the data window, but the screen center is generally an imaginary center line in the data display area.

Remarks:

7.13 DiagnosticStatus

Description:

Enumerates the states of power-on system diagnostics. The TPI method ITlaSystem.GetDiagnosticStatus returns a member of this enumeration.

Namespace: Tektronix.LogicAnalyzer.TpiNet

Declaration Syntax:

Visual Basic

```
Public Enum DiagnosticStatus As Byte
```

C#

```
public enum DiagnosticStatus : Byte
```

C++

```
__value public enum DiagnosticStatus : Byte
```

Enumeration Members:

`Unknown` – The diagnostic state of the system cannot be determined.

`Running` – System diagnostics are currently running.

`Passed` – The system has passed all diagnostics.

`Mixed –`

`Failed –`

7.14 CalibrationStatus

Description:

Enumerates the states of the system calibration. The TPI method ITlaSystem.GetCalibrationStatus returns a member of this enumeration.

Namespace: Tektronix.LogicAnalyzer.TpiNet

Declaration Syntax:

Visual Basic

```
Public Enum CalibrationStatus As Byte
```

C#

```
public enum CalibrationStatus : Byte
```

C++

```
__value public enum CalibrationStatus : Byte
```

Enumeration Members:

`Unknown` – The calibration status has not been determined.

`Calibrated` – The system is currently calibrated.

`Running` – The system is currently running the self-calibration procedure.

`Failed` – Self-calibration has failed.

`NotRequired` – No modules in the system require calibration.

7.15 TriggerResource

Description:

Enumerates the types of trigger resources associated with a data source.

Namespace: Tektronix.LogicAnalyzer.Common

Declaration Syntax:

Visual Basic

```
Public Enum TriggerResource As Byte
```

C#

```
public enum TriggerResource : Byte
```

C++

```
__value public enum TriggerResource : Byte
```

Enumeration Members:

State – Trigger state, treated as a resource
Clause – Trigger state clause, treated as a resource
Event – Trigger event resource
Action – Trigger action resource

Remarks:

All resource types can be thought of as having limited availability:

- Data sources usually have a finite number of trigger states. In the case of DSOs, they may have exactly 1 state.
- Trigger states usually have a finite number of trigger clauses. Again, in the case of DSOs, they may have exactly 1 clause.
- Events are resources whose logical state determines what trigger actions the data source performs. A trigger clause usually can support only a finite number of trigger events. Using an event resource often involves trade-offs with other types of events or actions.
- Actions are resources that cause a change of behavior in the data source at run time. Actions can be such things as triggering the data source, incrementing a counter, or asserting an output signal. A trigger clause usually can support only a finite number of trigger actions. Using an action resource often involves trade-offs with other types of actions or events.

7.16 TriggerConflict

Description:

Enumerates the types of trigger resource conflicts associated with a TlaTriggerResourceException.

Namespace: Tektronix.LogicAnalyzer.Common

Declaration Syntax:

Visual Basic

```
Public Enum TriggerConflict As UInt32
```

C#

```
public enum TriggerConflict: UInt32
```

C++

```
__value public enum TriggerConflict: UInt32
```

Enumeration Members:

NoneLeft – All resources of the requested type are in use already
OtherResource – Requested resource conflicts with another resource already in use
UnsupportedType – Data source's trigger interface doesn't support the requested resource
InvalidDefinition – Requested resource's definition is invalid
NoDelete – Requested resource cannot be removed

Remarks:

A trigger resource's definition can be invalid (InvalidDefinition) for any of several reasons:

- A non-trigger resource elsewhere in the data source that the trigger resource definition refers to can't be found.
- A setting in the resource's definition is out of bounds.
- The syntax of the resource's definition is illegal.

The NoDelete conflict occurs when the caller has requested that the last clause in a state or the last state in the trigger definition be removed.

7.17 LogicOperator

Description:

Enumerates the types of logic operators used for combining LA trigger events.

Namespace: Tektronix.LogicAnalyzer.Common

Declaration Syntax:

Visual Basic

```
Public Enum LogicOperator As Byte
```

C#

```
public enum LogicOperator : Byte
```

C++

```
__value public enum LogicOperator : Byte
```

Enumeration Members:

LogicalAnd – Logically AND event outputs
LogicalOr – Logically OR event outputs

Remarks:

None.

7.18 GlobalStorageType

Description:

Enumerates the types of trigger-controlled global storage behavior.

Namespace: Tektronix.LogicAnalyzer.Common

Declaration Syntax:

Visual Basic

```
Public Enum GlobalStorageType As Byte
```

C#

```
public enum GlobalStorageType : Byte
```

C++

```
__value public enum GlobalStorageType : Byte
```

Enumeration Members:

StoreAll – Store all acquired samples
StoreNone – Do not store any acquired samples
Transitional – Store when a monitored channel group's value changes
Conditional – Store based on LA trigger events
StartStop – Control storage based on Start and Stop actions

Remarks:

None.

7.19 StartStopStoreType

Description:

Enumerates the available start/stop global storage types.

Namespace: Tektronix.LogicAnalyzer.Common

Declaration Syntax:

Visual Basic

```
Public Enum StartStopStoreType As Byte
```

C#

```
public enum StartStopStoreType: Byte
```

C++

```
__value public enum StartStopStoreType: Byte
```

Enumeration Members:

StartStore – Store all acquired samples
StopStore – Do not store any acquired samples

Remarks:

None.

7.20 RunStateValue

Description:

The RunStateValue enumeration lists all acquisition run states for an acquisition module.

Namespace: Tektronix.LogicAnalyzer.TpiNet

Declaration Syntax:

Visual Basic

```
public Enum RunStateValue As Integer
```

C#

```
public enum RunStateValue : int
```

C++

```
__value public enum RunStateValue : int
```

Enumeration Members:

Idle – Module is not running and may be accessed for acquisition data.
Ready – Module is being programmed in preparation for acquisition.
Started – Module has been started but not yet armed.
Armed – Module is armed but not yet triggered.
Triggered – Module has triggered but memory not filled.
Complete – Module memory has filled but post acquisition cleanup is pending.

Remarks:

None.

Examples:

See IDetailedStatus.RunState for an example.

7.21 DataSetValue

Description:

The DataSetValue enumeration lists all data sets for which acquisition records may be requested.

Namespace: Tektronix.LogicAnalyzer.TpiNet

Declaration Syntax:

Visual Basic

```
public Enum DataSetValue As Integer
```

C#

```
public enum DataSetValue : int
```

C++

```
__value public enum DataSetValue : int
```

Enumeration Members:

Main – Main records.
MagniVu – MagniVu records.
Violation – Glitch Violation or Setup/Hold Violation records.
Invalid – Invalid bits records (for 2X or 4X LA acquisitions).
Compare – Compare differences records.

Remarks:

None.

Examples:

```
See the IRecordAccess.NumberOfRecords() for an example use of the
DataSetValue enumeration.
```

7.22 PolarityValue

Description:

The PolarityValue enumeration defines normal and inverted polarity for LA channels.

Namespace: Tektronix.LogicAnalyzer.TpiNet

Declaration Syntax:

Visual Basic

```
public Enum PolarityValue As Integer
```

C#

```
public enum PolarityValue : int
```

C++

```
__value public enum PolarityValue : int
```

Enumeration Members:

Positive
Negative

7.23 CompareRelativeValue

Description:

The CompareRelativeValue enumeration defines begin and trigger relative to values. These are used by the ComparePoint structure and ICompareSetup interface.

Namespace: Tektronix.LogicAnalyzer.TpiNet

Declaration Syntax:

Visual Basic

```
public Enum CompareRelativeValue As Integer
```

C#

```
public enum CompareRelativeValue : int
```

C++

```
__value public enum CompareRelativeValue : int
```

Enumeration Members:

RelativeToBegin
RelativeToTrigger

7.24 CompareStatusValue

Description:

The CompareStatusValue enumeration defines repetitive compare status values for the module. This is used in ILADetailedStatus.CompareStatus().

Namespace: Tektronix.LogicAnalyzer.TpiNet

Declaration Syntax:

Visual Basic

```
public Enum CompareStatusValue As Integer
```

C#

```
public enum CompareStatusValue : int
```

C++

```
__value public enum CompareStatusValue : int
```

Enumeration Members:

CompareDisabled – compare is disabled
CompareNoStatus – enabled but no status available (probably starting)
CompareRunning – compare is running
CompareEqual – compare found no differences
CompareNotEqual – compare found differences
CompareNoChannels – all channels were masked for compare
CompareNoSamples – no samples were available or reference alignment caused no overlap

7.25 AcquireModeValue

Description:

The AcquireModeValue enumeration defines acquire mode values for the LA module. These are used by the IAcquireMode interface.

Namespace: Tektronix.LogicAnalyzer.TpiNet

Declaration Syntax:

Visual Basic

```
public Enum AcquireModeValue As Integer
```

C#

```
public enum AcquireModeValue : int
```

C++

```
__value public enum AcquireModeValue : int
```

Enumeration Members:

AcquireModeNormal
AcquireModeBlocks
AcquireModeGlitches
AcquireModeSetupAndHold

7.26 ClockModeValue

The ClockModeValue enumeration defines clock mode values for the LA module. These are used by the IClockMode interface.

Namespace: Tektronix.LogicAnalyzer.TpiNet

Declaration Syntax:

Visual Basic

```
public Enum ClockModeValue As Integer
```

C#

```
public enum ClockModeValue : int
```

C++

```
__value public enum ClockModeValue : int
```

Enumeration Members:

ClockModeInternal
ClockModeInternal2X
ClockModeInternal4X
ClockModeExternal
ClockModeExternal2X
ClockModeExternal4X
ClockModeSourceSync
ClockModeCustom

7.27 StatusBitsModeValue

The StatusBitsModeValue enumeration defines values to represent how the status bits of the LA module are utilized during an acquisition.

Namespace: Tektronix.LogicAnalyzer.TpiNet

Declaration Syntax:

Visual Basic

```
public Enum StatusBitsModeValue As Integer
```

C#

```
public enum StatusBitsModeValue : int
```

C++

```
__value public enum StatusBitsModeValue : int
```

Enumeration Members:

CurrentTriggerState – status bits hold the trigger state that stored the sample
NextTriggerState – status bits hold the trigger state that was entered after the sample was stored
CurrentClockingState – status bits hold the clocking state that stored the sample
NextClockingState – status bits hold the clocking state that was entered after the sample was stored
ClockingGroupValid – status bits hold the clocking group valid bits for the sample

7.28 CounterTimerModeValue

The CounterTimerModeValue enumeration defines how the LA module is using a counter/timer resource.

Namespace: Tektronix.LogicAnalyzer.TpiNet

Declaration Syntax:

Visual Basic

```
public Enum CounterTimerModeValue As Integer
```

C#

```
public enum CounterTimerModeValue : int
```

C++

```
__value public enum CounterTimerModeValue : int
```

Enumeration Members:

Unused
Counter
Timer

7.29 TriggerReasonValue

The TriggerReasonValue enumeration defines values for what caused a module to trigger.

Namespace: Tektronix.LogicAnalyzer.TpiNet

Declaration Syntax:

Visual Basic

```
public Enum TriggerReasonValue As Integer
```

C#

```
public enum TriggerReasonValue : int
```

C++

```
__value public enum TriggerReasonValue : int
```

Enumeration Members:

NoTrigger – no trigger
SelfTrigger – module triggered itself independent of system trigger
SystemTrigger – system trigger triggered module
UnknownTrigger – indeterminate trigger (either self or system)

7.30 GroupRadix

Description:

Enumerates the radices that are applicable to logic analyzer groups.

Namespace: Tektronix.LogicAnalyzer.TpiNet

Declaration Syntax:

Visual Basic

```
Public Enum GroupRadix As Byte
```

C#

```
public enum GroupRadix : byte
```

C++

```
__value public enum GroupRadix : char
```

Enumeration Members:

```
Binary
Octal
Hex
UnsignedDecimal
SignedDecimal
Symbolic
```

7.31 FieldDelimiter

Enumerates the delimiters that are used in Import/Export Channel setups.

Namespace: Tektronix.LogicAnalyzer.TpiNet

Declaration Syntax:

Visual Basic

```
public Enum FieldDelimiter As Integer
```

C#

```
public enum FieldDelimiter : int
```

C++

```
__value public enum FieldDelimiter : int
```

Enumeration Members:

Tab
Comma
Space
Semicolon

8 Event Argument Classes

This section lists all specialized argument classes that are derived from System.EventArgs. These are classes that are passed as through the second argument of an event handler delegate. Only properties and methods that are in addition to those System.EventArgs are described in this section. See the MSDN library for a full description of the EventArgs base class.

8.1 RepetitiveStopEventArgs

Description:

An object of this class is passed to handlers of the ITlaRunControl.RepetitiveRunStopped event, and the object can be used to determine the reason why the most recent repetitive acquisition stopped.

Namespace: Tektronix.LogicAnalyzer.TpiNet

Declaration Syntax:

Visual Basic

```
Class RepetitiveStopEventArgs
      Inherits EventArgs
```

C#

```
class RepetitiveStopEventArgs : EventArgs
```

C++

```
__gc class RepetitiveStopEventArgs : public EventArgs
```

Remarks:

The purpose of this EventArgs-derived class is pass data about the repetitive stop reason to handlers of the ITlaRunControl.RepetitiveRunStopped event. The stop reason is contained in the Reason property.

8.1.1 RepetitiveStopEventArgs Properties

8.1.1.1 RepetitiveStopEventArgs.Reason

Description:

The value of this property is a member of the RepetitiveStopReason enumeration, which indicates the reason why the most recent repetitive acquisition stopped. This property is read only.

Declaration Syntax:

Visual Basic

```
Property Property-Name As Type-Name
```

C#

```
Type-Name Property-Name { set; get; }
```

C++

```
Type-Name* get_Property-Name ();
void set_Property-Name (Type_Name*);
```

Arguments:

Argument – Description of argument.

Exceptions Thrown:

None.

8.2 LockingWindowEventArgs

Description:

An object of this class is passed to handlers of the ILockingWindow.ItemChanged event, and

Namespace: Tektronix.LogicAnalyzer.TpiNet

Declaration Syntax:

Visual Basic

```
Class LockingWindowEventArgs
      Inherits EventArgs
```

C#

```
class LockingWindowEventArgs : EventArgs
```

C++

```
__gc class LockingWindowEventArgs : public EventArgs
```

8.2.1 LockingWindowEventArgs Properties

8.2.1.1 LockingWindowEventArgs.LockedItem

Description:

Specifies the item that was changed. The item is either Cursor1, Cursor2 or DisplayCenter.

Declaration Syntax:

Visual Basic

```
ReadOnly Property LockedItem As LockingWindowItem
```

C#

```
LockingWindowItem LockedItem { get; }
```

C++

```
LockingWindowItem get_LockedItem ();
```

Arguments:

None

Exceptions Thrown:

None.

8.3 SystemCollectionEventArgs

Description:

This class provides event data about a change to one of the ITlaSystem collections. When an ITlaSystem collection changes an event is fired such as ITlaSystem.DataSourcesChanged, a SystemCollectionEventArgs object is passed to the event handler to describe how the collection has changed.

Namespace: Tektronix.LogicAnalyzer.TpiNet

Declaration Syntax:

Visual Basic

```
Class SystemCollectionEventArgs
      Inherits EventArgs
```

C#

```
class SystemCollectionEventArgs : EventArgs
```

C++

```
__gc class SystemCollectionEventArgs : EventArgs
```

8.3.1 SystemCollectionEventArgs Properties

8.3.1.1 SystemCollectionEventArgs.Element

Description:

The value of this property is the element that has been added or removed from an ITlaSystem collection such as DataSources when the collection changed.

Declaration Syntax:

Visual Basic

```
ReadOnly Property Element As Object
```

C#

```
object Element { get; }
```

C++

```
Object* get_Element ();
```

Arguments:

None.

Exceptions Thrown:

None.

8.3.1.2 SystemCollectionEventArgs.IsElementAdded

Description:

This is a Boolean property that indicates whether the Element property represents an element that has been added to the collection. If IsElementAdded is true, then Element was added; and if it false, then Element was removed.

Declaration Syntax:

Visual Basic

```
ReadOnly Property IsElementAdded As Boolean
```

C#

```
bool IsElementAdded { get; }
```

C++

```
bool get_IsElementAdded ();
```

Arguments:

None.

Exceptions Thrown:

None.

8.4 RunStateEventArgs

Description:

When the run state of a data source changes, it raises an event that has this class as one of its arguments. The RunStateEventArgs class has a NewState property that specifies the new state that caused the event to be raised.

Namespace: Tektronix.LogicAnalyzer.Common

Declaration Syntax:

Visual Basic

```
Class RunStateEventArgs
      Inherits EventArgs
```

C#

```
interface RunStateEventArgs : EventArgs
```

C++

```
__gc __interface RunStateEventArgs : public EventArgs
```

8.4.1 RunStateEventArgs Properties

8.4.1.1 RunStateEventArgs.NewState

The value of this property is a member of the RunStateValue enumeration. The value specifies the run state of a data source at the time that an event was raised. The state is the one into which the data source changed.

Declaration Syntax:

Visual Basic

```
ReadOnly Property NewState As RunStateValue
```

C#

```
RunStateValue NewState { get; }
```

C++

```
RunStateValue get_NewState ();
void set_NewState (RunStateValue );
```

Arguments:

None.

Exceptions Thrown:

None

8.5 SymbolFileChangedArgs

Description:

This class contains arguments passed to a event handler when the ITlaSystem object notifies clients that a symbol table has been loaded or unloaded.

Namespace: Tektronix.LogicAnalyzer.TpiNet

Declaration Syntax:

Visual Basic

```
Class SymbolFileChangedArgs
      Inherits EventArgs
```

C#

```
class SymbolFileChangedArgs: EventArgs
```

C++

```
__gc class SymbolFileChangedArgs: public EventArgs
```

Remarks:

The SymbolFile property of this class is set by the ITlaSystem object to indicate which symbol file has been loaded or unloaded.

8.5.1 SymbolFileChangedArgs Properties

8.5.1.1SymbolFileChangedArgs.SymbolFile

Description:

The value of this property is the full path name of a symbol file that has been loaded or unloaded .

Declaration Syntax:

Visual Basic

```
ReadOnly Property SymbolFile As String
```

C#

```
string SymbolFile { get; }
```

C++

```
String* get_SymbolFile ();
```

Arguments:

None.

Exceptions Thrown:

None.

8.6 ObjectEventArgs

Description:

This event data class is the second argument to events that defined using the ObjectEventHandler delegate. The event data of this class allows event handlers to distinguish between the sender of event and the object that was the ultimate cause of the event.

Namespace: Tektronix.LogicAnalyzer.TpiNet

Declaration Syntax:

Visual Basic

```
Class ObjectEventArgs
      Inherits EventArgs
```

C#

```
class ObjectEventArgs : EventArgs
```

C++

```
__gc class ObjectEventArgs : EventArgs
```

Remarks:

The Source property references the object that changed to cause the event.

8.6.1 ObjectEventArgs Properties

8.6.1.1 ObjectEventArgs.Source

Description:

The value of this property is a reference to the object that changed to cause an event to be raised. This is useful when several objects can change to cause the same event.

Declaration Syntax:

Visual Basic

```
ReadOnly Property Source As Object
```

C#

```
object Source { get; }
```

C++

```
Object* get_Source ();
```

Arguments:

None.

Exceptions Thrown:

None.

Remarks:

The value of this property is set by an argument to the constructor.

8.6.2 ObjectEventArgs Methods

8.6.2.1 ObjectEventArgs Constructor

Description:

Creates an ObjectEventArgs objects with property value specified by the constructor argument.

Declaration Syntax:

Visual Basic

```
Sub New (sourceObject As Object)
```

C#

```
ObjectEventArgs (object sourceObject )
```

C++

```
ObjectEventArgs (Object* sourceObject )
```

Arguments:

`sourceObject` – This object is used to initialize the Source property.

Return Value:

None.

Exceptions Thrown:

None.

9 Delegates

This section lists all specialized delegates used by the interfaces in this document.

9.1 RepetitiveStopHandler

Description:

This delegate is used by objects that subscribe to ITlaRunControl.RepetitiveRunStopped. Event handlers that are represented by this delegate take an argument that indicates why the repetitive run has finished.

Namespace: Tektronix.LogicAnalyzer.TpiNet

Declaration Syntax:

Visual Basic

```
<Serializable>
Delegate Sub RepetitiveStopHandler _
      (sender As Ojbect, e As RepetitiveStopEventArgs)
```

C#

```
[Serializable]
delegate void RepetitiveStopHandler
      (object sender, RepetitiveStopEventArgs e)
```

C++

```
[Serializable]
__delegate Type-Name* RepetitiveStopHandler
      (Object* sender, RepetitiveStopEventArgs* e)
```

Arguments:

`sender` – The object that raised the event.

`e` – A RepetitiveStopEventArgs object that contains event data.

Return Value:

None.

9.2 LockedItemChangedHandler

Description:

Objects that subscribe to events raised by ILockingWindow objects use this delegate. The delegate defines a parameter that indicates which item within a locking window caused the event.

Namespace: Tektronix.LogicAnalyzer

Declaration Syntax:

Visual Basic

```
Delegate Sub LockedItemChangedHandler _
      (sender As Ojbect, e As LockingWindowEventArgs)
```

C#

```
delegate void LockedItemChangedHandler (
      object sender, LockingWindowEventArgs e)
```

C++

```
__delegate void LockedItemChangedHandler
      (Object* sender, LockingWIndowEventArgs* e)
```

Arguments:

`sender` – The object that raised the event.

`e` – A `LockingWIndowEventArgs` object that contains event data.

Return Value:

None

Remarks:

ILockingWindow objects raise events having the signature of this delegate. Handlers of locking window events can use the property LockingWindowsEventArgs.LockedItem to determine which locked item in the window changed and caused the event.

9.3 CollectionChangedHandler

Description:

This delegate is used for events that are fired when an ITlaSystem collection such as ITlaSystem.DataSources changes. Objects subscribe to those events with delegates of this type.

Namespace: Tektronix.LogicAnalyzer.TpiNet

Declaration Syntax:

Visual Basic

```
Delegate Sub CollectionChangedHandler _
      (sender As Object, e As SystemCollectionEventArgs)
```

C#

```
delegate void CollectionChangedHandler
      (object sender, SystemCollectionEventArgs e)
```

C++

```
__delegate void CollectionChangedHandler
      (object* sender, SystemCollectionEventArgs* e)
```

Arguments:

`sender` – The object that raised the event.

`e` – A `SystemCollectionEventArgs` object that contains event data.

Return Value:

Methods represented by delegates of this kind do not return a value.

Remarks:

ITlaSystem collection change events, such as DataWindowChanged, are fired when elements are added or removed from the collection. The e argument can be used to determine what element caused the change and whether the element was added or removed.

9.4 RunStateChangedHandler

Description:

This is the delegate used by events that are raised when the run state of a data source has changed.

Namespace: Tektronix.LogicAnalyzer.Common

Declaration Syntax:

Visual Basic

```
Delegate Sub RunStateChangedHandler _
      (sender As Object, e As RunStateEventArgs)
```

C#

```
delegate void RunStateChangedHandler
      (Object sender, RunStateEventArgs e)
```

C++

```
__delegate void RunStateChangedHandler
      (Object* sender, RunStateEventArgs* e)
```

9.5 SymbolFileChangedHandler

Description:

Delegates of this type are used to subscribe to events that are raised when symbol files are loaded or unloaded.

Namespace: Tektronix.LogicAnalyzer.TpiNet

Declaration Syntax:

Visual Basic

```
<Serializable>
Delegate Sub SymbolFileChangedHandler _
      (sender As Ojbect, e As SymbolFileChangedArgs)
```

C#

```
[Serializable]
delegate void SymbolFileChangedHandler
      (object sender, SymbolFileChangedArgs e)
```

C++

```
[Serializable]
__delegate void SymbolFileChangedHandler
      (Object* sender, SymbolFileChangedArgs* e)
```

Arguments:

`sender` – The object that raised the event.

`e` – A `SymbolFileChangedArgs` object that contains the name of the symbol file that changed.

Return Value:

None.

9.6 ObjectEventHandler

Description:

This delegate may be used to subscribe to a variety of events. The object that changed to cause the event is passed as event data. This delegate is useful for events in which an object raises an event because one of several sub-objects has changed.

Namespace: Tektronix.LogicAnalyzer.TpiNet

Declaration Syntax:

Visual Basic

```
Delegate Sub ObjectEventHandler (sender As Object, _
                                 e as ObjectEventArgs)
```

C#

```
delegate void ObjectEventHandler (object sender, ObjectEventArgs e)
```

C++

```
__gc __delegate void ObjectEventHandler (Object* sender,
                                         ObjectEventArgs e)
```

Arguments:

`sender` – The object that raised the event.

`e` – An ObjectEventArgs object that contains a reference to the object that changed to cause the event. That object is not necessarily the same as `sender`, it might be a sub-object of `sender`.

Return Value:

None.

10 Plug-In Support Classes

This section describes classes that are implemented in TlaNetInterfaces.dll to support plug-ins and plug-in development.

10.1 PlugInIdentityAttribute

Description:

A class that directly or indirectly implements IPlugIn applies this attribute to specify how the TLA application should identify the plug-in to users. This primary used of this attribute is to specify a user-friendly name for the plug-in. This name will appear in the TLA UI in places where the user can create instances of plug-ins, for example in the Tools menu or in the New Data Window wizard.

Additionally, this attribute his two named parameters that allow a plug-in class to specify an associated icon and an group name to which to the plug-in name belongs.

Namespace: Tektronix.LogicAnalyzer.PlugIns

Declaration Syntax:

Visual Basic

```
<AttributeUsage (AttributeTargets.Class)>
Class PlugInIdentityAttribute
      Inherits Attribute
```

C#

```
[AttributeUsage (AttributeTargets.Class)]
class PlugInIdentityAttribute : Attribute
```

C++

```
[AttributeUsage (AttributeTargets.Class)]
__gc class PlugInIdentityAttribute : Attribute
```

Remarks:

PlugInIdentityAttribute targets classes, which allows multiple plug-ins to be packaged in the same assembly while allowing unique names to be associated with each plug-in.

Plug-in classes are not required to apply this attribute. However, if a plug-in implementation does not apply this attribute, the class indicates that users should not be able to instantiate it from the TLA user interface; and the TLA will omit the plug-in from all menus, toolbars, or wizards that identify available plug-ins. Such a plug-in can only become part of the system if created programmatically, either by another plug-in or by an external TPI.NET client.

10.1.1 PlugInIdentityAttribute Properties

10.1.1.1 PlugInIdentityAttribute.Name

Description:

Gets the name of the name of the plug-in as it should appear to users. The TLA uses this property to identify Tool plug-ins in the Tools menu, Data Window Plug-ins in the New Data Window wizard, and Data Source Plug-ins in the Add Data Source dialog.

Declaration Syntax:

Visual Basic

```
ReadOnly Property Name As String
```

C#

```
string Name { get; }
```

C++

```
String* get_Name ();
```

Arguments:

None.

Exceptions Thrown:

This property does not throw exceptions.

Remarks:

This is generally a short name like “Graph Window” that quickly identifies the plug-in. The name should be specific enough to distinguish it from other plug-ins that might be added. If the TLA encounters two plug-ins of the same kind with the same name, it will append “ n” where n is a number. For example two data source plug-ins named “LA Emulation” would be appear as “LA Emulation 1” and “LA Emulation 2.”

The Name property is set through the positional parameter of PlugInIdentityAttribute. Any plug-in implementation that uses the attribute must specify a string parameter for the name.

Examples:
The following examples declare a data window plug-in named "LA Math Window."

Visual Basic

```
<PlugInIdentity("LA Math Window")>
Public Class MathWindow
      Inherits IDataWindowPlugIn
```

C#

```
[PlugInIdentity("LA Math Window")]
public class MathWindow : IDataWindowPlugIn
```

10.1.1.2 PlugInIdentityAttribute.IconFileName

Description:

Sets or gets the name of the an icon. IPlugIn class declarations can use this attribute to indicate that they have an associated icon. This icon will be used by the TLA application in places where the plug-in can be instantiated from the TLA UI.

Declaration Syntax:

Visual Basic

```
Property IconFileName As String
```

C#

```
string IconFileName { set; get; }
```

C++

```
String* get_IconFileName ();
void set_IconFileName (String*);
```

Arguments:

None.

Exceptions Thrown:

This property does not throw exceptions.

Remarks:

The property is the named parameter that specifies the simple name of the icon, for example name.ico. The full path of the icon file is not needed because the file must be in the same directory as the DLL that contains the class implementation.

Examples:

To declaratively set this property, a plug-in class declaration initializes the IconFileName named parameter as shown below. In the examples two tools are declared, and both will be grouped under an "Acme Products" sub-menu of the Tools menu. Each tool will have a different icon next to its menu item because different icon files are specified.

Visual Basic

```
<PlugInIdentity("TLA Tool", _
      GroupName:="Acme Products", IconFileName:="tool1.ico")>
Public Class TlaTool
      Inherits IPlugIn

<PlugInIdentity("Analysis Tool", _
      GroupName:="Acme Products", IconFileName:="tool2.ico")>
Public Class AnalysisTool
```

```
      Inherits IPlugIn
```

C#

```
[PlugInIdentity("TLA Tool", GroupName:="Acme Products",
      IconFileName:="tool1.ico")>
Public Class TlaTool : IPlugIn

<PlugInIdentity("Analysis Tool",
      GroupName:="Acme Products", IconFileName:="tool2.ico")>
Public Class AnalysisTool : IPlugIn
```

10.1.1.3 PlugInIdentityAttribute.GroupName

Description:

Gets or sets a group name to which a plug-in belongs.

The TLA normally adds a menu item for an IPlugIn implementation directly into the Tools menu. If the GroupName property is set and is not empty, then the TLA will add the menu item for the plug-in to a menu named GroupName, which will be a submenu of the Tools menu. This allows plug-ins to be logically grouped together in the Tools menu.

Declaration Syntax:

Visual Basic

```
Property GroupName As String
```

C#

```
string GroupName { set; get; }
```

C++

```
String* get_GroupName ();
void set_GroupName  (String*);
```

Arguments:

None.

Exceptions Thrown:

This property does not throw exceptions.

Examples:

To declaratively set this property, a plug-in class declaration initializes the GroupName named parameter as shown below. In the examples two tools are declared, and both will be grouped under an "Acme Products" sub-menu of the Tools menu.

Visual Basic

```
<PlugInIdentity("TLA Tool", GroupName:="Acme Products")>
Public Class TlaTool
      Inherits IPlugIn

<PlugInIdentity("Analysis Tool", GroupName:="Acme Products")>
Public Class AnalysisTool
      Inherits IPlugIn
```

C#

```
[PlugInIdentity("TLA Tool", GroupName = "Acme Products")]
Public Class TlaTool: IPlugIn
```

```
[PlugInIdentity("Analysis Tool", GroupName = "Acme Products")]
Public Class AnalysisTool: IPlugIn
```

10.1.1.4 PlugInIdentityAttribute.GroupIconFileName

Description:

This is the file name for an icon associated with the group to which the plug-in belongs.

Declaration Syntax:

Visual Basic

```
Property GroupIconFileName As String
```

C#

```
string GroupIconFileName { set; get; }
```

C++

```
String* get_GroupIconFileName ();
void set_GroupIconFileName(String*);
```

Arguments:

None.

Exceptions Thrown:

None.

10.1.2 PlugInIdentityAttribute Methods

10.1.2.1 PlugInIdentityAttribute Constructor

Description:

This attribute has one constructor, which takes a string argument to initialize the name property.

Declaration Syntax:

Visual Basic

```
Sub new (name As String)
```

C#

```
PlugInIdentityAttribute(string name)
```

C++

```
PlugInIdentityAttribute(string name);
```

Arguments:

`name` – A string argument for the user-friendly name of the plug-in. The attribute Name property is set to this value.

Return Value:

None

Exceptions Thrown:

ArgumentNullException :
Condition: The `name` argument is null.
Message: Cannot construct PlugInIdentityAttribute from a null reference.

Remarks:

This constructor is used by compilers to generate metadata associated with the PlugInIdentityAttribute custom attribute.

Examples:
The following examples declare a data window plug-in named "LA Math Window." The name string is passed to the attribute constructor and becomes metadata associated with the class.

Visual Basic

```
<PlugInIdentity("LA Math Window")>
Public Class MathWindow
      Inherits IDataWindowPlugIn
```

C#

```
[PlugInIdentity("LA Math Window")]
public class MathWindow : IDataWindowPlugIn
```

10.2 PlugInInstantiationAttribute

Description:

Class implementations that derive from IPlugIn use this attribute to control how the TLA instantiates plug-ins. PlugInInstantiationAttribute controls whether the TLA automatically creates an instance at startup and whether the TLA will allow more than one instance of the plug-in to be created.

Namespace: Tektronix.LogicAnalyzer.PlugIns

Declaration Syntax:

Visual Basic

```
<AttributeUsage (AttributeTargets.Class)>
Class PlugInInstantiationAttribute
      Inherits Attribute
```

C#

```
[AttributeUsage (AttributeTargets.Class)]
class PlugInInstantiationAttribute: Attribute
```

C++

```
[AttributeUsage (AttributeTargets.Class)]
__gc class PlugInInstantiationAttribute: public Attribute
```

Remarks:

This class has two properties that are set through constructor parameters (unnamed attribute parameters when used declaratively). The IsSingleInstance property controls whether the TLA is constrained to create no more than one instance of a given plug-in class. The IsInstantiatedAtStartUp property controls whether the plug-in is automatically created when the application starts.

Although the TLA application does not require this attribute, it is good practice to decorate all plug-in class implementations with PlugInInstantiationAttribute. Since multiple instance plug-ins are expected to be more common that single instance plug-ins, the TLA application will allow multiple instances of plug-in classes that lack this attribute.

This attribute is only meaningful for classes that implement IPlugIn.

10.2.1 PlugInInstantiationAttribute Properties

10.2.1.1 PlugInInstantiationAttribute.IsSingleInstance

Description:

This is a Boolean property that indicates whether or not the TLA application may create multiple instances of a class that derives from IPlugIn. When the value of this property is true, the TLA application will never create more than one instance of the class.

Declaration Syntax:

Visual Basic

```
ReadOnly Property IsSingleInstance As Boolean
```

C#

```
bool IsSingleInstance{ get; }
```

C++

```
bool get_IsSingleInstance();
```

Arguments:

None.

Exceptions Thrown:

This property does not throw exceptions.

Remarks:

Whenever an IPlugIn implementation is decorated with PlugInInstantiationAttribute, the value of this property is checked before instantiating the class; and only one instance at a time will existing in the system when the value is true.

The value of this property is set in the first parameter of the PlugInInstantiationAttribute constructor.

10.2.1.2 PlugInInstantiationAttribute.IsInstantiatedAtStartUp

Description:

The value of this property controls whether or not an instance of the plug-in is instantiated when the TLA application starts up.

Declaration Syntax:

Visual Basic

```
ReadOnly Property IsInstantiatedAtStartUp As Boolean
```

C#

```
bool IsInstantiatedAtStartUp { get; }
```

C++

```
bool get_IsInstantiatedAtStartUp ();
```

Arguments:

None.

Exceptions Thrown:

None.

Remarks:

If this value is true, then the TLA will instantiate one instance of the plug-in at application start-up. If both IsInstantiatedAtStartUp and IsSingleInstance have true values, then the class is a single instance plug-in whose sole instance is created at start-up, and neither users nor TPI.NET clients will be able to create new instances unless the existing instance is deleted.

The value of this property is set in the second parameter of the PlugInInstantiationAttribute constructor.

10.2.2 PlugInInstantiationAttribute Methods

10.2.2.1 PlugInInstantiationAttribute Constructor

Description:

The constructor of takes two parameters. The first indicates whether the class is limited to a single instance or can be multiply instantiated. The second parameter indicates whether or not an instance of the class should be created as part of a default system.

Declaration Syntax:

Visual Basic

```
Sub New (isSingleInstance As Boolean, autoCreate As Boolean)
```

C#

```
PlugInInstantiationAttribute(bool isSingleInstance, bool autoCreate);
```

C++

```
PlugInInstantiationAttribute(bool isSingleInstance, bool autoCreate);
```

Arguments:

`isSingleInstance` – This argument sets the IsSingleInstance property. If set to true, the attribute indicates that the TLA must not attempt to create multiple instances of the class decorated by the attribute.

autoCreate – This argument sets the IsInstatiatedAtStartUp property. If set to true, the application will automatically create an instance of the IPlugIn class when the application starts up.

Return Value:

Not applicable

Exceptions Thrown:

This constructor does not throw exceptions.

Remarks:

Specific details about the method.

Examples:

C#

The following code declares a plug-in class of which the TLA application will create at most one instance, and the plug-in will not be automatically instantiated.

```
[PlugInInstantiation(true, false)]
public class CustomLATool : IPlugIn { ... }
```

10.3 PlugInSavedInfo

Description:

This class provides access to the setup data of a saved plug-in instance within a .tla file. The access is provided through a SerializationInfo object. PlugInSavedInfo exposes the SerializationInfo object through its 'Info' property. Data values saved in the file can be deserialized using the SerializationInfo methods in the same manner as they would be used inside a deserialization constructor.

PlugInSavedInfo objects are obtained by calling ITlaPlugInSupport.GetSavedData(). In order to successfully used the SerializationInfo object provided PlugInSavedInfo, you must know the names of the values that have been serialized by a plug-in instance. In general, this means that plug-ins can meaningfully use a PlugInSavedInfo object only if it represents the saved data from a plug-in of the same type.

Namespace: Tektronix.LogicAnalyzer.TpiNet.

Declaration Syntax:

Visual Basic

```
Class PlugInSavedInfo
      Inherits IDisposable
```

C#

```
public class PlugInSavedInfo : IDisposable
```

C++

```
public __gc class PlugInSavedInfo : public IDisposable
```

10.3.1 PlugInSavedInfo Properties

10.3.1.1 PlugInSavedInfo.Info

Description:

The value of this property is an object of type SerializationInfo. This object can be used to deserialize the data values of a plug-in setup that has been saved into a .tla file.

Declaration Syntax:

Visual Basic

```
Property Info As SerializationInfo
```

C#

```
SerializationInfo Info { get; }
```

C++

```
SerializationInfo* get_Info ();
```

Arguments:

None.

Exceptions Thrown:

None.

Remarks:

The SerializationInfo object is connected to saved data through a Stream object. That stream object is the value of the InfoStream object.

10.3.1.2 PlugInSavedInfo.InfoStream

Description:

The value of this property is the stream that is used get data out of the file. In most cases, a plug-in client will not need to directly access this stream.

Declaration Syntax:

Visual Basic

```
Property InfoStream As Stream
```

C#

```
Stream InfoStream { get; }
```

C++

```
Stream* get_InfoStream ();
```

Arguments:

None.

Exceptions Thrown:

None.

Remarks:

When PlugInSavedInfo.Dispose is called, this stream is closed.

10.3.2 PlugInSavedInfo Methods

10.3.2.1 PlugInSavedInfo.Dispose

Description:

This method implements the IDisposable interface. When called, it frees all unmanaged resources by closing the file stream through which the SerializationInfo object access saved data.

Declaration Syntax:

Visual Basic

```
Sub Dispose ()
```

C#

```
void Dispose ()
```

C++

```
void Dispose ()
```

Arguments:

None.

Return Value:

None.

Exceptions Thrown:

None.

Remarks:

When a PlugInSavedInfo object is no longer needed, this Dispose() method must be called to prevent waste of limited operating system resources.

10.4 TlaForm

Description:

This class is a type of Form window that has been specialized to work well as a child of the TLA main application window. TlaForm is provided as a base class for plug-in developers who need to derive Form windows for user interfaces. If a plug-in calls ITlaPlugInSupport.TopLevelFormToChild() to put a Form inside the main application window, better UI integration is achieved when the child Form is derived from TlaForm.

Namespace: Tektronix.LogicAnalyzer.TpiNet

Declaration Syntax:

Visual Basic

```
Class TlaForm
      Inherits Form
```

C#

```
class TlaForm : Form
```

C++

```
__gc class TlaForm : public Form
```

Remarks:

Classes derived from TlaForm are visually identical to classes derived from Form. TlaForm overrides the following Form methods to make them work as expected within the context of the TLA application:

Overridden Form methods:

```
      Activate()
      BringToFront()
      Hide()
      SendToBack();
      Show();
```

Overridden Form property:

```
      WindowsState
```

TlaForm provides two properties that are not part of the Form base class. These properties, HelpID and HelpFile, when properly set, enable the Help menu item "Help on Window."

10.4.1 TlaForm Properties

10.4.1.1 TlaForm.HelpID

Description:

If the TlaForm window has been made a child of the main application window, the value of this property is the help topic ID associated with the window. The default value of this property is zero, which indicates that window does not have a help topic. If this value is set to zero, the "Help on Window" menu item is disabled when the window has the focus. If this value is greater than zero, the "Help on Window" item is enabled.

TlaForm.HelpId and TlaForm.HelpFile together determine what topic to display when the user selects "Help on Window". The help topic is displayed by invoking WinHelp. HelpFile determines the help file to open, and HelpID determines the topic to show within that file.

Declaration Syntax:

Visual Basic

```
Property HelpId me As Integer
```

C#

```
int HelpId { set; get; }
```

C++

```
int get_HelpId ();
void set_HelpId (int);
```

Arguments:

None.

Exceptions Thrown:

None.

Remarks:

Unless the HelpID is set, "Help on Window" in the Help menu is disabled.

10.4.1.2 TlaForm.HelpFile

Description:

If the TlaForm window has been made a child of the main application window, the value of this property is the help file associated with the window. This is the file that is opened when the window has the focus and the user selects "Help on Window" from the Help menu.

The default value of this property is an empty string. An empty string indicates that the standard TLA application help file, TLA700.hlp, should be opened when "Help on Window" is selected.

TlaForm.HelpId and TlaForm.HelpFile together determine what topic to display when the user selects "Help on Window". The help topic is displayed by invoking WinHelp. HelpFile determines the help file to open, and HelpID determines the topic to show within that file.

Declaration Syntax:

Visual Basic

```
Property HelpId me As Integer
```

C#

```
int HelpId { set; get; }
```

C++

```
int get_HelpId ();
void set_HelpId (int);
```

Arguments:

None.

Exceptions Thrown:

None.

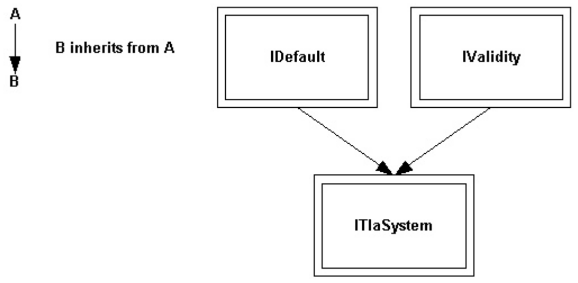
Remarks:

If the value of TlaForm.HelpID is not greater than zero, then the "Help on Window" command is disabled. In that case, the value of HelpFile is not used.

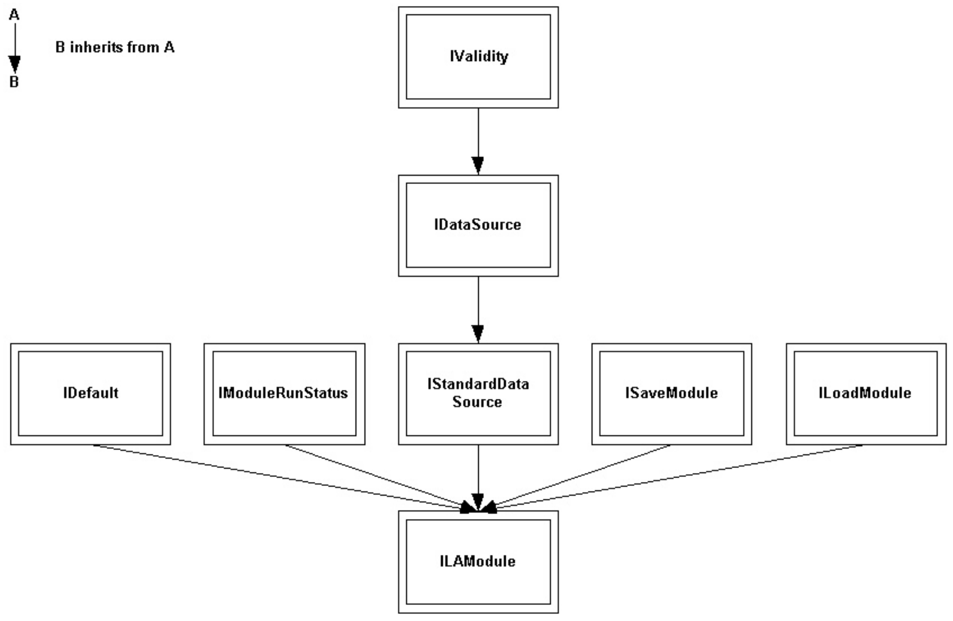
11 Appendices

11.1 Appendix A: Selected Interface Inheritance Diagrams

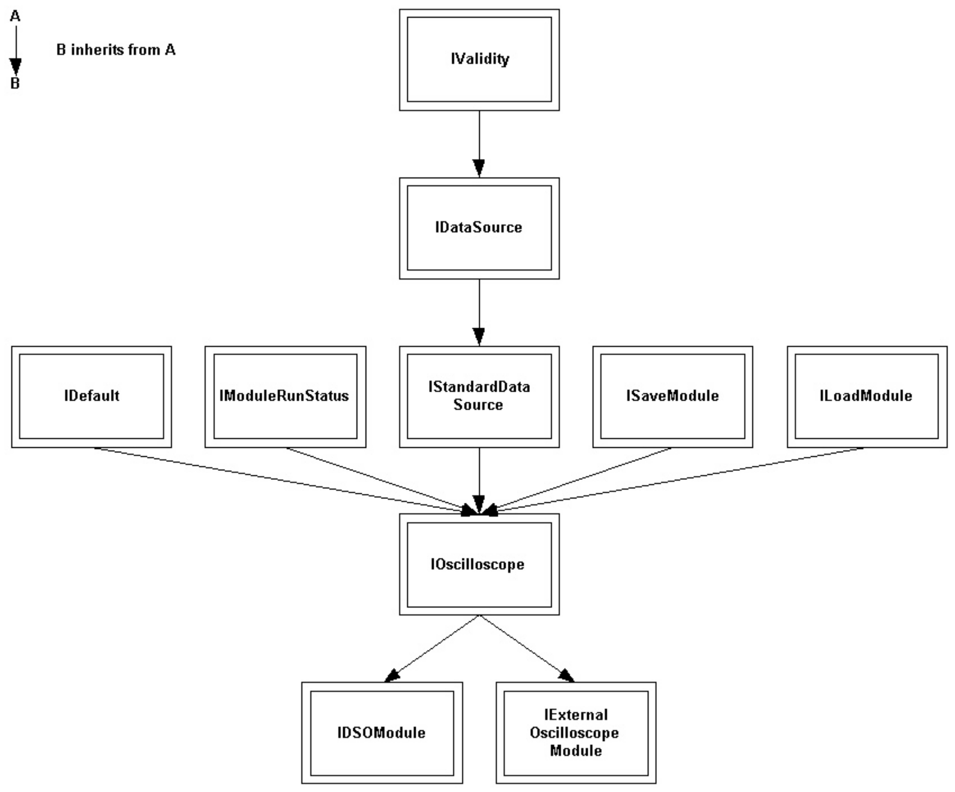
11.1.1 ITlaSystem



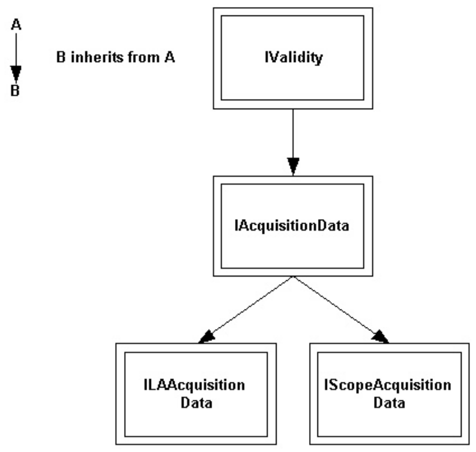
11.1.2 ILAModule



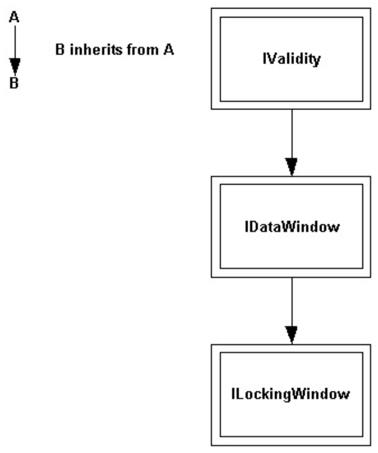
11.1.3 IDSOModule and IExternalOscilloscopeModule



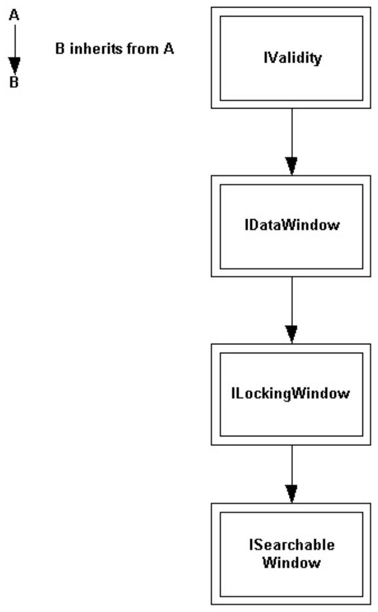
11.1.4 ILAAcquisitionData and IScopeAcquisitionData



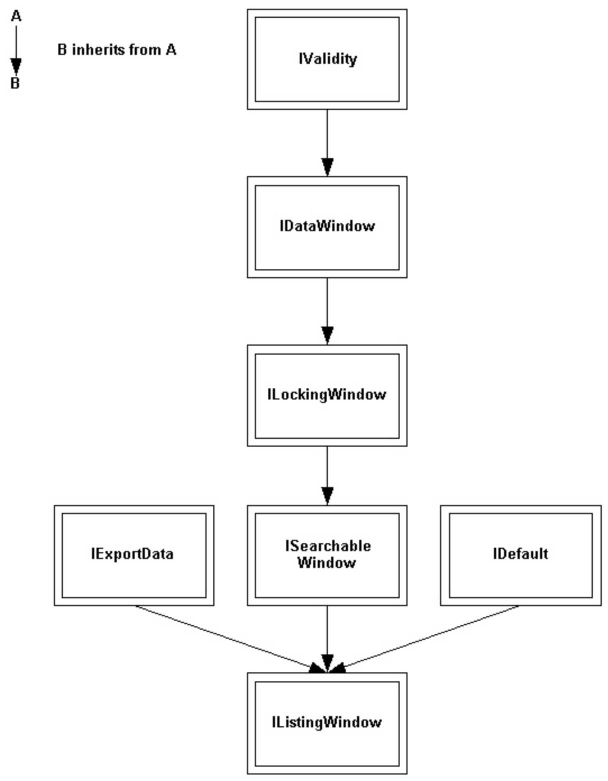
11.1.5 ILockingWindow



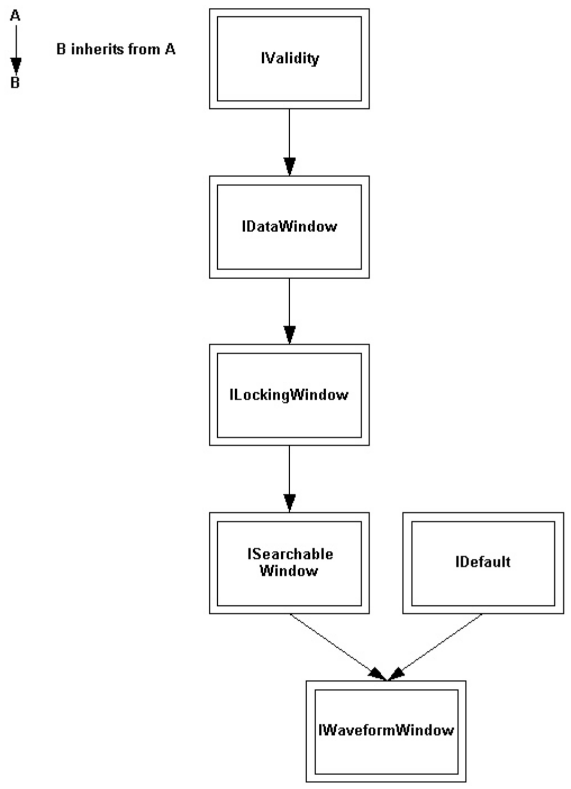
11.1.6 ISearchableWindow



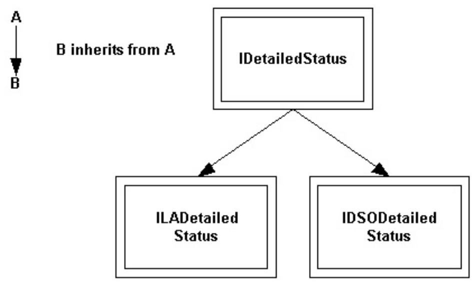
11.1.7 IListingWindow



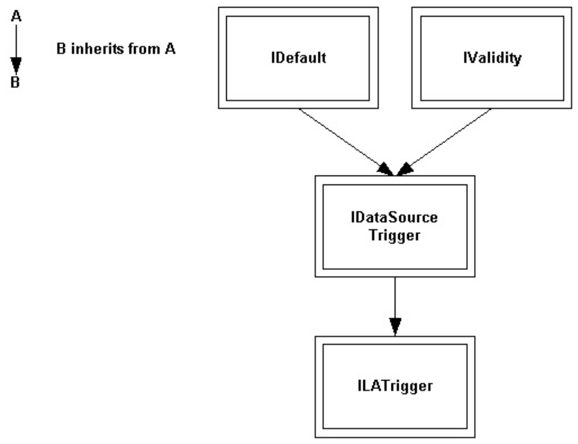
11.1.8 IWaveformWindow



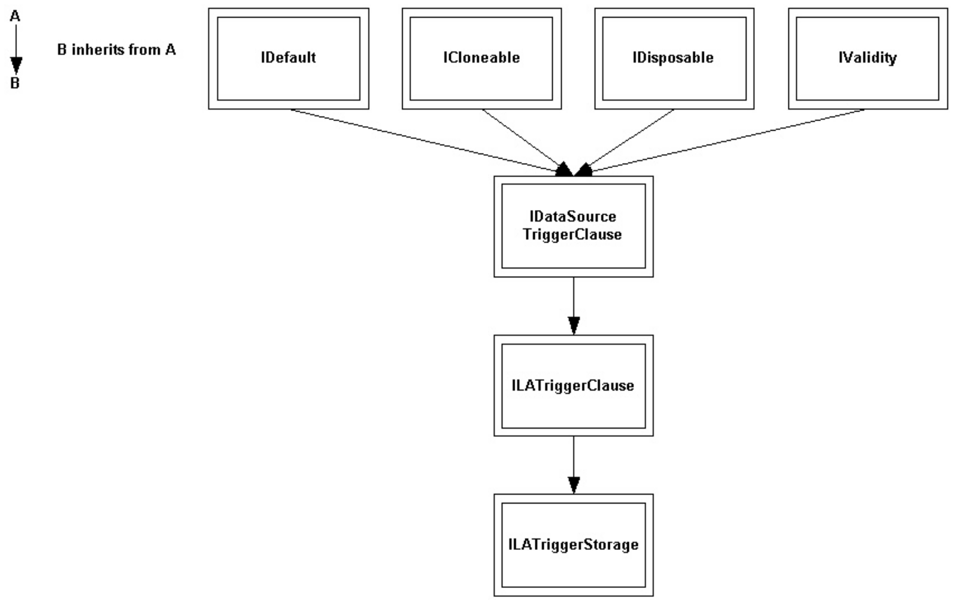
11.1.9 ILADetailedStatus and IDSODetailedStatus



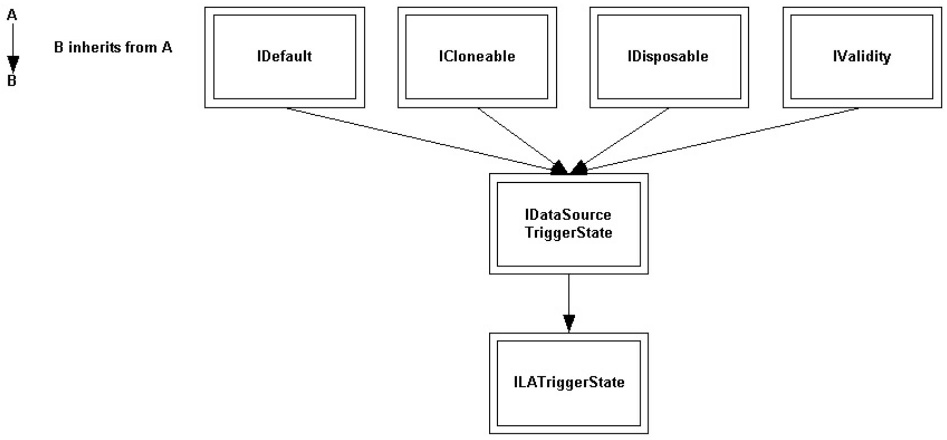
11.1.10 ILATrigger



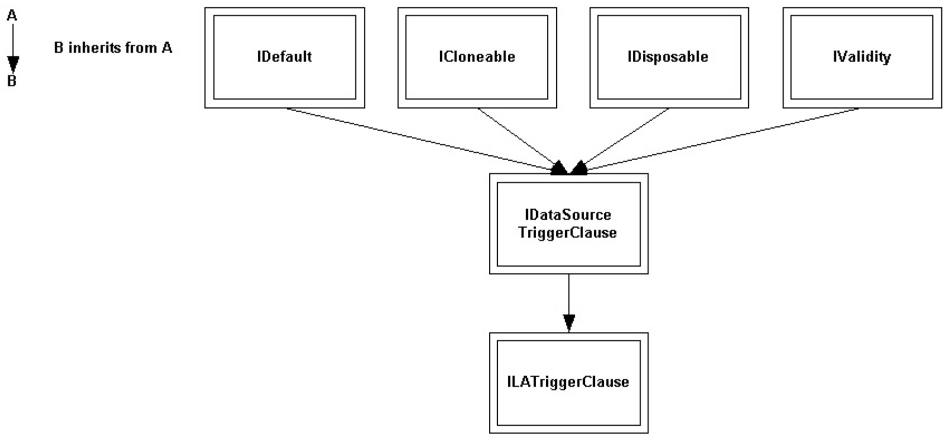
11.1.11 ILATriggerStorage



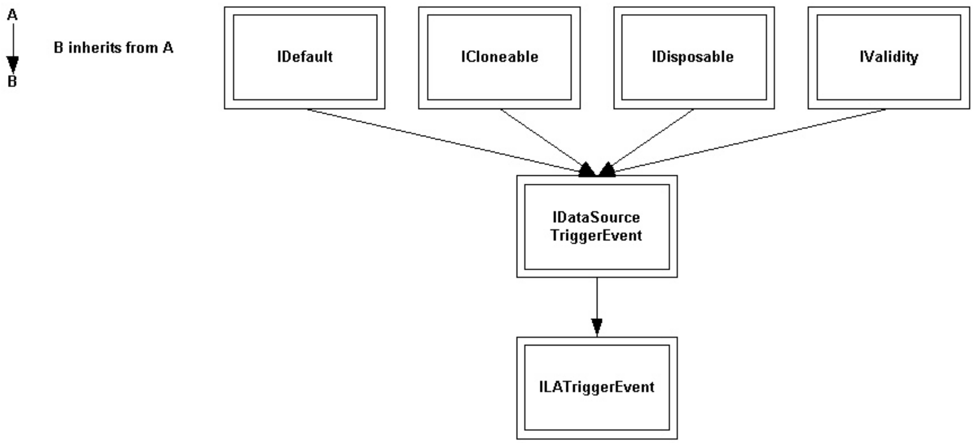
11.1.12 ILATriggerState



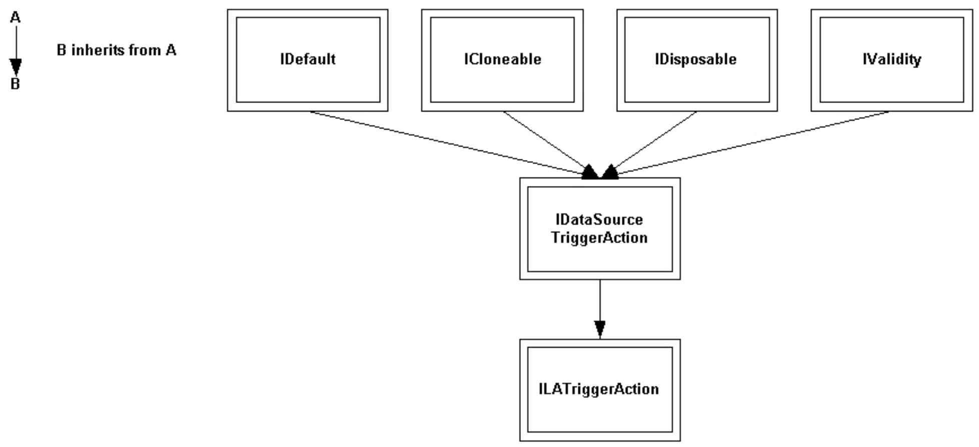
11.1.13 ILATriggerClause



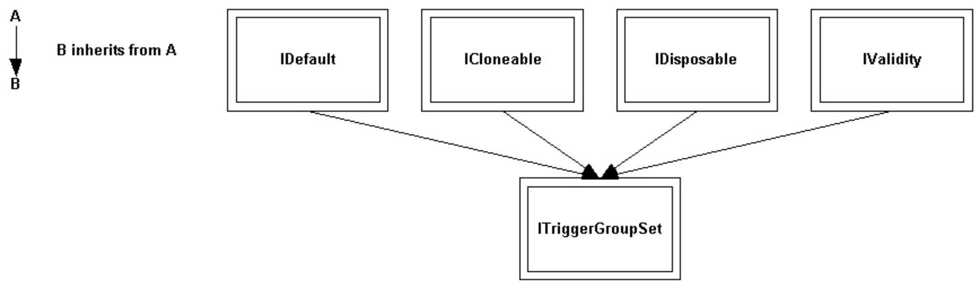
11.1.14 ILATriggerEvent



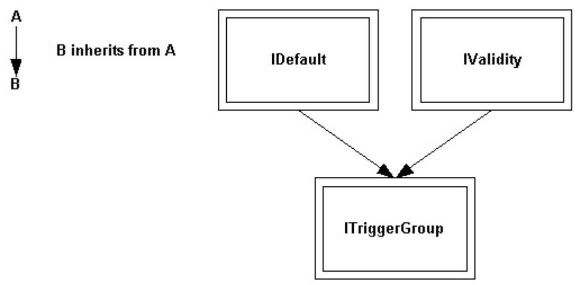
11.1.15 ILATriggerAction



11.1.16 ILATriggerGroupSet



11.1.17 ILATriggerGroup



Appendix B: Raw Record Formats

Merged LA Data Format

A merged module combines data from all modules into each data record. The data record for a merged module consists of the data for the master module immediately followed by data from the slave modules (Master followed by Slave 1 followed by Slave 2, etc.). Data for the slave modules never contains RTC, reserved, or timestamp bytes (same as MagniVu). For example, for two merged 7L4's, a main data record consists of 43 bytes: 26 bytes of E3(7-0) through Timestamp(7-0) for the master module, followed by 17 bytes of E3(7-0) through Clock/Quals for the slave module (see the figure below). A MagniVu data record for the same module would be 9 bytes narrower due to absence of the RTC, reserved, and timestamp bytes.

E3 master 7 6 5 4 3 2 1 0	E2 master 7 6 5 4 3 2 1 0	E1 master 7 6 5 4 3 2 1 0	E0 master 7 6 5 4 3 2 1 0
A3 master 7 6 5 4 3 2 1 0	A2 master 7 6 5 4 3 2 1 0	D3 master 7 6 5 4 3 2 1 0	D2 master 7 6 5 4 3 2 1 0
A1 master 7 6 5 4 3 2 1 0	A0 master 7 6 5 4 3 2 1 0	D1 master 7 6 5 4 3 2 1 0	D0 master 7 6 5 4 3 2 1 0
C3 master 7 6 5 4 3 2 1 0	C2 master 7 6 5 4 3 2 1 0	C1 master 7 6 5 4 3 2 1 0	C0 master 7 6 5 4 3 2 1 0
Clock/Quals master	RTC Bits 7-0	Reserved Bits 7-0	Timestamp 5 pad + 50-48
Timestamp 47-40	Timestamp 39-32	Timestamp 31-24	Timestamp 23-16
Timestamp 15-8	Timestamp 7-0	E3 slave 7 6 5 4 3 2 1 0	E2 slave 7 6 5 4 3 2 1 0
E1 slave 7 6 5 4 3 2 1 0	E0 slave 7 6 5 4 3 2 1 0	A3 slave 7 6 5 4 3 2 1 0	A2 slave 7 6 5 4 3 2 1 0
D3 slave 7 6 5 4 3 2 1 0	D2 slave 7 6 5 4 3 2 1 0	A1 slave 7 6 5 4 3 2 1 0	A0 slave 7 6 5 4 3 2 1 0
D1 slave 7 6 5 4 3 2 1 0	D0 slave 7 6 5 4 3 2 1 0	C3 slave 7 6 5 4 3 2 1 0	C2 slave 7 6 5 4 3 2 1 0
C1 slave 7 6 5 4 3 2 1 0	C0 slave 7 6 5 4 3 2 1 0	Clock/Quals slave	

See later sections for additional master/slave record formats.

LA RTC Data

Run time control data for the LA is available in the RTC byte of the acquisition vector. This byte is only valid for the main data set of the LA (it is not valid for the violation data set or the MagniVu data set), however the RTC data byte of the main data set can be applied to the corresponding sample of the violation data set if violation data was acquired.

Bits 7-4 of the RTC data form a status nibble. The status nibble may contain the following type of data:
TSM State – trigger state machine state during which the sample was stored. Trigger states are zero based. All LA module types support storage of this type of data in the status nibble.
CSM State – clocking state machine state number during which the sample was stored. Clocking states are zero based. Only TLA7Axx modules are capable of storing this type of data in the status nibble.
CGC Data – clock group complete data with bits for each clock group. Only TLA7Axx modules are capable of storing this type of data in the status nibble. Bit 7 of RTC is CGC3, bit 6 of RTC is CGC 2, bit 5 of RTC is CGC 1, and bit 4 of RTC is CGC 0.

Bit 3 of the RTC data is the gap bit. A value of 1 indicates that one or more samples of data immediately prior to that sample were not stored.

Bits 2-0 of the RTC data are reserved.

LA Timestamps
LA timestamps are only present in the master or standalone module. LA timestamps are not present in data for slave modules or in the MagniVu data set. LA timestamps are zero filled for invalid and compare data sets. The timestamp value represents the time at which the sample was stored relative to starting the module acquisition. To convert a timestamp into a time value, multiply it by the picosecond tick rate from IRecordAccess.RawTimestampMultiplier.

136-Channel LA (7L4 / 7M4 / 7N4 / 7P4 / 7Q4 / 7Ax4 / 6*X*4 / 5XX4) Record Format
A data record for the master (or standalone) module consists of 26 bytes, starting with E3(7-0) and ending with Timestamp(7-0). A MagniVu record (or additional slave module bytes) consist of 17 bytes, starting with E3(7-0) and ending with Clock/Quals.

E3 7 6 5 4 3 2 1 0	E2 7 6 5 4 3 2 1 0	E1 7 6 5 4 3 2 1 0	E0 7 6 5 4 3 2 1 0
A3 7 6 5 4 3 2 1 0	A2 7 6 5 4 3 2 1 0	D3 7 6 5 4 3 2 1 0	D2 7 6 5 4 3 2 1 0
A1 7 6 5 4 3 2 1 0	A0 7 6 5 4 3 2 1 0	D1 7 6 5 4 3 2 1 0	D0 7 6 5 4 3 2 1 0
C3 7 6 5 4 3 2 1 0	C2 7 6 5 4 3 2 1 0	C1 7 6 5 4 3 2 1 0	C0 7 6 5 4 3 2 1 0
Clock/Quals (See ordering)	RTC Bits 7-0	Reserved Bits 7-0	Timestamp 5 pad + 50-48
Timestamp 47-40	Timestamp 39-32	Timestamp 31-24	Timestamp 23-16
Timestamp 15-8	Timestamp 7-0		

Clock / Qualifier Byte:

7 Q3	6 Q2	5 Q1	4 Q0	3 CK3	2 CK2	1 CK1	0 CK0

102-Channel LA (7L3 / 7M3 / 7N3 / 7P3 / 7Ax3 / 6*X*3 / 5XX3) Record Format
A data record for the master (or standalone) module consists of 22 bytes, starting with A3(7-0) and ending with Timestamp(7-0). A MagniVu record (or additional slave module bytes) consist of 13 bytes, starting with A3(7-0) and ending with Clock/Quals.

A3 7 6 5 4 3 2 1 0	A2 7 6 5 4 3 2 1 0	D3 7 6 5 4 3 2 1 0	D2 7 6 5 4 3 2 1 0
A1 7 6 5 4 3 2 1 0	A0 7 6 5 4 3 2 1 0	D1 7 6 5 4 3 2 1 0	D0 7 6 5 4 3 2 1 0
C3 7 6 5 4 3 2 1 0	C2 7 6 5 4 3 2 1 0	C1 7 6 5 4 3 2 1 0	C0 7 6 5 4 3 2 1 0
Clock/Quals (See ordering)	RTC Bits 7-0	Reserved Bits 7-0	Timestamp 5 pad + 50-48
Timestamp 47-40	Timestamp 39-32	Timestamp 31-24	Timestamp 23-16
Timestamp 15-8	Timestamp 7-0		

Clock / Qualifier Byte:

7	6	5	4	3	2	1	0
Pad	Pad	Q1	Q0	CK3	CK2	CK1	CK0

68-Channel LA (7L2 / 7M2 / 7N2 / 7P2 / 7Q2 / 7Ax2 / 6*X*2 / 5XX2) Record Format
A data record for the module consists of 18 bytes, starting with A1(7-0) and ending with Timestamp(7-0). A MagniVu record consists of 9 bytes, starting with A1(7-0) and ending with Clock/Quals.

A1 7 6 5 4 3 2 1 0	A0 7 6 5 4 3 2 1 0	D1 7 6 5 4 3 2 1 0	D0 7 6 5 4 3 2 1 0
C3 7 6 5 4 3 2 1 0	C2 7 6 5 4 3 2 1 0	A3 7 6 5 4 3 2 1 0	A2 7 6 5 4 3 2 1 0
Clock/Quals (See ordering)	RTC Bits 7-0	Reserved Bits 7-0	Timestamp 5 pad + 50-48
Timestamp 47-40	Timestamp 39-32	Timestamp 31-24	Timestamp 23-16
Timestamp 15-8	Timestamp 7-0		

Clock / Qualifier Byte:

7	6	5	4	3	2	1	0
Pad	Pad	Pad	Pad	CK3	CK2	CK1	CK0

34-Channel LA (7L1 / 7M1 / 7N1 / 7P1 / 7Ax1 / 6*X*1 / 5XX1) Record Format
A data record for the module consists of 14 bytes, starting with C3(7-0) and ending with Timestamp(7-0). A MagniVu record consists of 5 bytes, starting with C3(7-0) and ending with Clock/Quals.

C3 7 6 5 4 3 2 1 0	C2 7 6 5 4 3 2 1 0	A3 7 6 5 4 3 2 1 0	A2 7 6 5 4 3 2 1 0
Clock/Quals (See ordering)	RTC Bits 7-0	Reserved Bits 7-0	Timestamp 5 pad + 50-48
Timestamp 47-40	Timestamp 39-32	Timestamp 31-24	Timestamp 23-16
Timestamp 15-8	Timestamp 7-0		

Clock / Qualifier Byte:

7	6	5	4	3	2	1	0
Pad	Pad	Pad	Pad	CK3	Pad	Pad	CK0

Interpretation of DSO Data
One single DSO channel's sample consists of a 16-bit short word in two's compliment signed form. Interpretation of this number as a voltage depends upon the channel's offset and range settings. A zero value for the DSO data word maps to the offset voltage.

Here is a summary of values returned by the DSO.

MSB LSB	
7FFF	Over range
7E00	Top of requested range (= offset voltage + ½ range)
0002	
0001	
0000	Mid-point for signed numbers (= offset voltage)
FFFF	
FFFE	
8200	Bottom of requested range (= offset voltage – ½ range)
8001	Under range
8000	No sample taken, or null point

Record Format – All 4 Channel Scope models

The record contains four 16-bit short words, one for each of the four channels, beginning with Ch1 and ending with Ch4. The 16-bit words are in Intel byte order.

Ch1 LSB 7 … 0	Ch1 MSB 7 … 0	Ch2 LSB 7 … 0	Ch2 MSB 7 … 0	Ch3 LSB 7 … 0	Ch3 MSB 7 … 0	Ch4 LSB 7 … 0	Ch4 MSB 7 … 0

Record Format – All 2 Channel Scope models

The record contains two 16-bit short words, one for each of the two channels, beginning with Ch1 and ending with Ch2. The 16-bit words are in Intel byte order.

Ch1 LSB 7 … 0	Ch1 MSB 7 … 0	Ch2 LSB 7 … 0	Ch2 MSB 7 … 0

Appendix C: Known TPI.NET Issues in V4.3 TLA Software

Issue 1:

IRecordAccess does not properly limit which samples are compared when the DataSet property is set to DataSetValue.Compare. It may return indications of compare differences on samples that are not being compared against the reference data source. Requests for compare samples should be limited to those samples that are actually being compared. ILAModule.GetCompareSetupObject() can be used to determine which samples are being compared.