TestStand ActiveX API Reference

TestStand ActiveX API Overview
- **ActiveX Automation Server Overview** 2
  - Classes, Objects, Methods, and Properties
  - Object References
- **TestStand ActiveX Server Overview** 3
  - Static and Dynamic Properties
  - Using the PropertyObject Class to Access Dynamic Properties
  - Accessing Objects, Properties, and Variables from Code Modules
  - Accessing TestStand Objects Inside Other TestStand Objects
  - Accessing Dynamic Properties in TestStand Objects
- **Using the TestStand ActiveX API in Different Programming Languages** 7
  - LabVIEW
  - LabWindows/CVI
  - Visual Basic
  - MFC/C++
- **Advanced Issues** 10
  - Adding and Releasing References
  - Thread Concurrency Models

Writing an Application 12

Class Method and Property Tree 23

Classes, Constants, and Enumerations 31

Important Information 213

Warranty, Medical Warning, Trademarks

National Instruments Technical Support

Press F1 for Help on Using Help

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

How to Print This Help File

To print topics from this help file perform the following steps:

1. Click on the **Contents** button in the button bar. The Help Topics dialog box appears.
2. In the Contents tab, select the book icon, , or topic icon, , that corresponds to the information that you want to print.
3. Click on the **Print** button.

TestStand ActiveX API Overview

This help file describes the basics of using an ActiveX automation server and, more specifically, the TestStand ActiveX automation server API. You can use the TestStand API to access the internal data of TestStand sequences and steps or to create and control sequence executions.

This help file assumes that you have read and are familiar with the concepts in *Getting Started with TestStand*, and Chapter 1, *TestStand Architecture Overview*, of the *TestStand User Manual*.

ActiveX Automation Server Overview

Classes, Objects, Methods, and Properties

Object References

An ActiveX server is any executable code that makes itself available to other applications according to the ActiveX standard. ActiveX implies a client/server relationship in which the client requests objects from the server and asks the server to perform actions on the objects.

Clients and servers can run on a single computer, or they can run on separate computers connected by a network. An ActiveX server is called *local* when it runs on the same computer as its client, and *remote* when it runs on a different computer.

On a single computer, an ActiveX server runs either *in-process* or *out-of-process* with respect to the client. An in-process ActiveX server is a dynamic-link library that runs in the same process space as the client. An out-of-process ActiveX server is an executable and thus does not run in the same process space as the client. Using an out-of-process server instead of an in-process server can add significant performance overhead to the client application.

Classes, Objects, Methods, and Properties

The interface for an ActiveX server consists of the following hierarchical, object-oriented categories:

Object -- An *object* is a service that an ActiveX server makes available to clients.

Class -- a *class* is like a data type definition except that it applies to objects rather than variables. A class defines a list of methods and properties that you can use with respect to the objects that you create as instances of that class. For some classes, you can create objects directly. For other classes, you can obtain objects only by calling a method or property on an already existing object of another class.

Method -- A *method* performs an operation or function on an object.

Property -- A *property* stores and maintains a setting or attribute of an object.

Object References

All ActiveX objects maintain reference counts. In other words, each object keeps track of the number of things that reference it. This allows the object to decide when to free the resources it uses. In most cases, test developers do not have to take any specific actions to maintain this count. Nevertheless, you have to help maintain the reference count if you call a method or property that returns an object or if you store a pointer to an object in a global variable for use after your test module completes. See the Advanced Issues topic for more information on references to ActiveX objects.

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

TestStand ActiveX Server Overview

Static and Dynamic Properties

Using the PropertyObject Class to Access Dynamic Properties

Identifying Properties when Calling PropertyObject Class Methods

Accessing Objects, Properties, and Variables from Code Modules

Accessing TestStand Objects Inside Other TestStand Objects

Accessing Dynamic Properties in TestStand Objects

Accessing Static Properties of an Object You Obtain as a Dynamic Property

The following are the classes in the TestStand API.

`PropertyObject`	`User`	`InteractiveContext`
`Engine`	`Adapter`	`InteractiveArgs`
`SequenceFile`	`Execution`	`EditArgs`
`Sequence`	`Thread`	`UIMessage`
`Step`	`SequenceContext`	`Report`

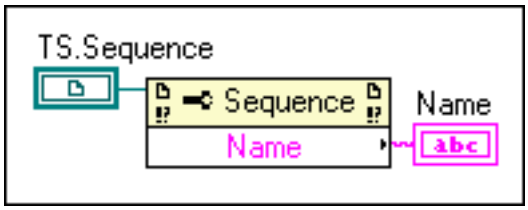
The `PropertyObject` class allows test developers to access sequence variables and custom step properties from step modules.

The other classes are necessary for developing a run-time operator interface. The operator interfaces that come with TestStand make extensive use of these classes. Of these classes, the `Engine` class is the only class for which you can create an object directly. The `Engine` class has methods for creating objects for the other classes. In some cases, you can obtain objects for one class from another class. For example, you can obtain a step object from the sequence object that contains the step.

Static and Dynamic Properties

TestStand defines a number of built-in properties that are always present for objects such as steps and sequences. Almost every kind of object in TestStand has built-in properties. The built-in properties are *static* with respect to the Application Programming Interface (API). In other words, the TestStand API has knowledge about each of these properties. The API employs this knowledge to make it easy for you to access these properties in the programming language you use.

In LabVIEW, you access the built-in properties through the property node. The following block diagram obtains the value of the `Name` property from a sequence object.



In C or C++, the TestStand API has methods for getting and setting the values of the built-in properties for each TestStand class. For example, the `Sequence` class has the `SequenceGetProperty` and `SequenceSetProperty` methods. When you call these methods in C or C++, you pass defined constants to identify specific properties. The following C code obtains the value of the `Name` property from a sequence object.

```
TS_SequenceGetProperty (sequence, &errorInfo, TS_SequenceName, CAVT_CSTRING,
&nameString);
```

The following Visual Basic code obtains the value of the `Name` property from a sequence object.

```
NameString = Sequence.Name
```

In Visual Basic, you access the built-in properties by using the native Visual Basic syntax for accessing properties of objects and controls.

TestStand also allows you to define your own custom step properties, sequence local variables, sequence file global variables, and station global variables. Because the API has no knowledge of the variables and custom step properties that you define, these variables and properties are *dynamic* with respect to the API. The TestStand API provides the `PropertyObject` class so that test developers can access dynamic properties and variables from within step modules. Rather than using defined constants, you use lookup strings to identify specific properties by name.

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

Using the PropertyObject Class to Access Dynamic Properties

Identifying Properties when Calling PropertyObject Class Methods

This topic describes when and how you use the `PropertyObject` class of the TestStand API.

You use the `PropertyObject` class to access dynamic properties. Typically, you use the `PropertyObject` class in step modules to access custom step properties, sequence local variables, sequence file variables, and station global variables. For most step modules that use the TestStand API, the `PropertyObject` class is the only TestStand API class that is necessary. Run-time operator interfaces, on the other hand, use all the TestStand API classes.

Objects of every class other than the `Engine` class can have custom properties. To access the custom properties of an object, you must use the methods in the `PropertyObject` class. To use methods in the `PropertyObject` class, you must have a PropertyObject reference for the object. For every class other than the `PropertyObject` and `Engine` classes, the TestStand API contains an `AsPropertyObject` method. The `AsPropertyObject` method converts the reference for an object of a specific class to a reference that is of the `PropertyObject` class but that refers to the same object. When you call the `AsPropertyObject` method, it increments the reference count for the object. You must release the object reference when you are done with it to decrement the reference count.

In most programming languages, you do not have to call the `AsPropertyObject` method. In LabWindows/CVI, the object references for all TestStand API classes have the same data type, `CAObjHandle`. Thus, you can pass an object handle for one class to a method for another class. Visual Basic allows you to assign object handles of one class directly to object handles of another class. In C++, you can associate the dispatch pointer for an object to an MFC wrapper for a different class. It is generally easier to use these programming language features rather than the `AsPropertyObject` method, because you do not create an extra object handle that you have to release later.

Because LabVIEW enforces strict typing, it is best to use the `AsPropertyObject` method. TestStand comes with a set of LabVIEW VIs that simplify getting and setting values of custom properties. Each VI calls the `AsPropertyObject` method, gets or sets a property value of a particular type, and then releases the PropertyObject reference. The VIs appear in the TestStand subpalette of the User Libraries palette of the LabVIEW Functions palette.

Identifying Properties when Calling PropertyObject Class Methods

When you call a `PropertyObject` class method, you can specify that it operates on the object itself or one of its subproperties. To specify what a method operates on, you pass a string that defines a complete path from the object on which you call the method to the specific property you want to access. This string is called a lookupString. To specify the object itself, pass an empty string (""). To specify a subproperty, pass the name of the subproperty. To specify a subproperty of a subproperty, pass a string that contains both names separated by a period(.). For example, you can specify the error code property in a step object that uses the following string:

```
"Result.Error.Code"
```

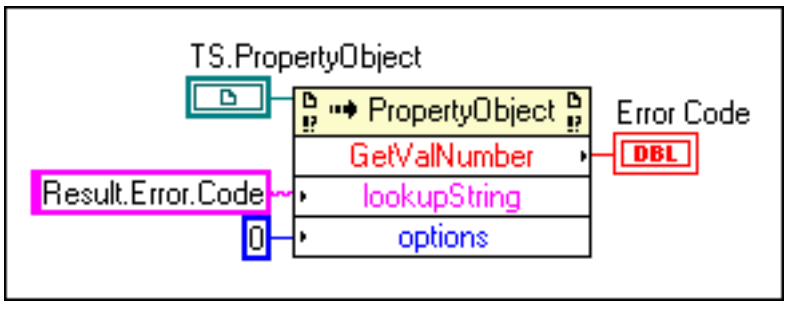
The complete LabWindows/CVI function call to obtain the error code property is:

```
ERRORINFO errorInfo;
double errorCode;

TS_PropertyGetValNumber (step, &errorInfo, "Result.Error.Code", 0, &errorCode);
```

The errorInfo variable is a structure that the LabWindows/CVI ActiveX Automation Library defines to hold information about errors that might occur in the operation of the function. The errorCode variable receives the error code property value from the step. The data type of the errorCode is double because TestStand stores all numeric values as 64-bit floating-point numbers.

The equivalent LabVIEW code is as follows:



© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

Accessing Objects, Properties, and Variables from Code Modules

Multiple ways exist for you to access objects, custom properties, and variables from code modules, depending on the module adapter that you use.

If you use the Flexible DLL Prototype Adapter for your step module, you can pass any number of parameters to the DLL function. The DLLs functions can have a variety of data types. TestStand translates values between TestStand data types and the data types that the DLL uses. For example, TestStand can translate a numeric value to a 64-bit floating point value, a 32-bit floating point value, a 32-bit signed integer value, a 32-bit unsigned integer value, and so on. If you want to pass a complete object, such as a step object or a variable with a custom data type, you must pass it as an automation object. You can pass an automation object as either an ActiveX Automation Dispatch Pointer or as a LabWindows/CVI Automation Handle. In the code module, you can use the TestStand ActiveX API to call methods on the automation object. You use the `PropertyObject` class to access values of variables, custom step properties, or their subproperties. You use the other TestStand API classes to call methods that are specific to objects such steps and sequences and to access the values of their built-in properties.

If you use the LabVIEW Standard Prototype Adapter or the C/CVI Standard Prototype Adapter for your step module, TestStand automatically passes a set of predefined parameters to your LabVIEW VI or C function. These parameters contain error information, status information, measurement information, and a *sequence context*, which is an object of the `SequenceContext` class. You cannot pass additional parameters. You can, however, use the sequence context to access all the objects, variables, and properties in the execution. To access the values of variables or custom step properties, you call `PropertyObject` class methods on the sequence context. Refer to Using the PropertyObject Class for information about calling `PropertyObject` methods on objects of the various TestStand API classes. For information on accessing objects such as steps and sequences and their built-in properties from the sequence context, refer to the following topic:

Accessing TestStand Objects Inside Other TestStand Objects

TestStand objects often contain references to other TestStand objects. For example, a sequence object contains references to the step objects for the steps that the sequence contains. A sequence context object contains references to many TestStand objects, which in turn reference all the TestStand objects and variables in an execution.

You access TestStand objects inside other TestStand objects differently depending on whether the objects exists as static properties in the TestStand API. For example, the `SequenceContext` class has a static property that represents the currently executing sequence. So you can use the static property to obtain a reference to the sequence. On the other hand, to obtain a particular step in a sequence, you must call the `GetStep` method in the `Sequence` class. For example, the following C code obtains the run mode string for of the third step in the Main step group of the currently executing sequence:

```
ERRORINFO errorInfo;
CAObjHandle sequence, step;
char *runModeString;

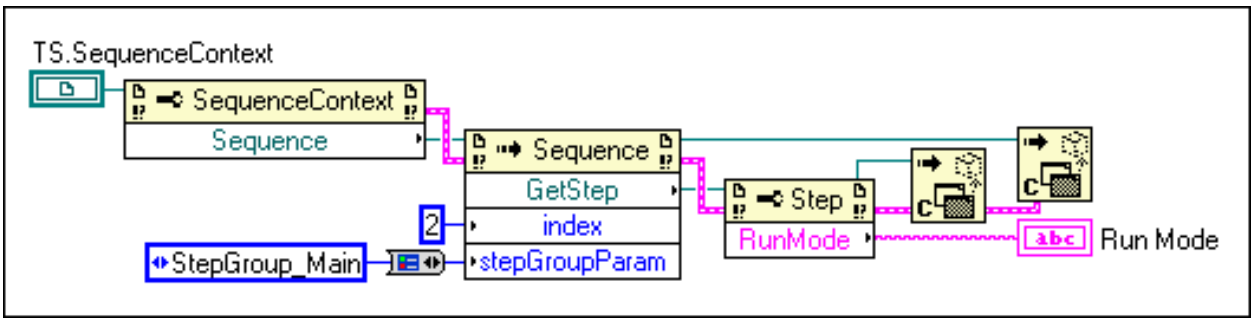
TS_SeqContextGetProperty (testData->seqContextCVI, &errorInfo,
TS_SeqContextSequence, CAVT_OBJHANDLE, &sequence);

TS_SequenceGetStep (sequence, &errorInfo, 2, TS_StepGroup_Main, &step);

TS_StepGetProperty (step, &errorInfo, TS_StepRunMode, CAVT_CSTRING, &runModeString);

CA_DiscardObjHandle (step);
CA_DiscardObjHandle (sequence);
```

The equivalent LabVIEW code is as follows:



© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

Accessing Dynamic Properties in TestStand Objects

Accessing Static Properties of an Object You Obtain as a Dynamic Property

TestStand objects can contain dynamic properties. For example, the step type for a step can have dynamic properties such as high and low limits. To access dynamic properties in a TestStand object, you have two alternatives. You can first use TestStand API methods or properties to obtain the reference to the object that has the property you want to access. Refer to Accessing TestStand Objects Inside Other TestStand Objects for details. You can then use a `PropertyObject` class method to access the dynamic property in the object. For example, the following C code gets the value of the low limit in the currently executing step.

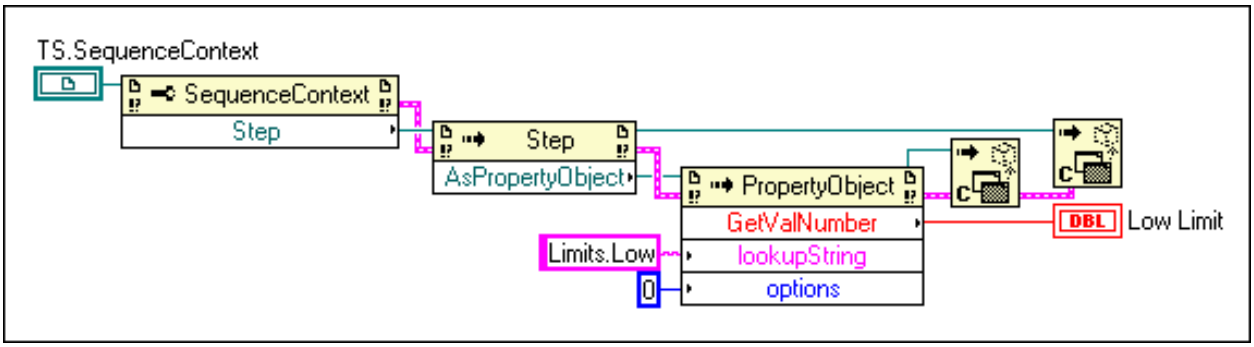
```
ERRORINFO errorInfo;
CAObjHandle step;
double lowLimit;

TS_SeqContextGetProperty (testData->seqContextCVI, &errorInfo, TS_SeqContextStep,
CAVT_OBJHANDLE, &step);

TS_PropertyGetValNumber (step, &errorInfo, "Limits.Low", 0, &lowLimit);

CA_DiscardObjHandle (step);
```

The equivalent LabVIEW code is as follows:

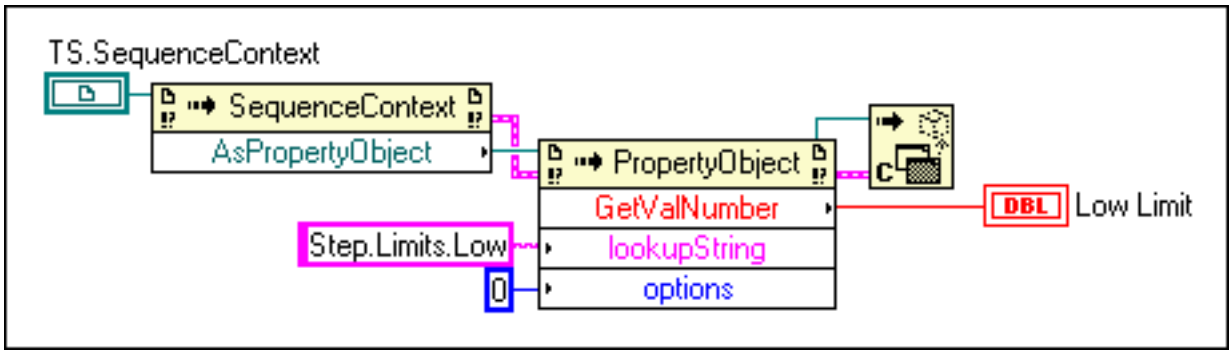


On the other hand, you can avoid obtaining the intermediate object reference by calling a `PropertyObject` method with a string that specifies a complete path from the top-level object to the dynamic property. For example, you can use the following code to replace the entire previous example:

```
ERRORINFO errorInfo;
double lowLimit;

TS_PropertyGetValNumber (testData->seqContextCVI, &errorInfo, "Step.Limits.Low", 0,
&lowLimit);
```

The equivalent LabVIEW code is as follows:



You can specify a complete path regardless of the length or complexity of the path, as long all the elements in the path are dynamic properties. For example, to specify the error code in the third step of the Main step group of the currently executing sequence, use the following path:

```
"RunState.Sequence.Main[2].Result.Error.Code"
```

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

Accessing Static Properties of an Object You Obtain as a Dynamic Property

In some cases you might obtain a reference to a TestStand object using the `PropertyObject` class methods and then want to access one of the static properties of the object. For example, you might want to obtain the run mode for the third step of the Main step group of the currently executing sequence. The example Accessing TestStand Objects Inside Other TestStand Objects shows how you can do this using methods in the `SequenceContext`, `Sequence` and `Step` classes. You can simplify the code in that example by using the `PropertyObject` class to obtain the PropertyObject reference for the step directly from the sequence context.

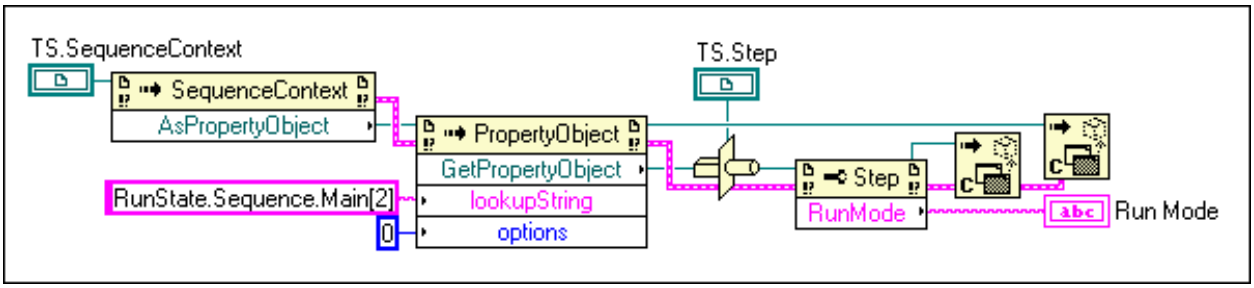
```
ERRORINFO errorInfo;
CAObjHandle step;
char *runModeString;

TS_PropertyGetPropertyObject (testData->seqContextCVI, &errorInfo,
"RunState.Sequence.Main[2]", 0, &step);

TS_StepGetProperty (step, &errorInfo, TS_StepRunMode, CAVT_CSTRING, &runModeString);

CA_DiscardObjHandle (step);
```

The LabVIEW code is as follows:



Notice that the class method `GetValObject` reference to access static properties in the `Step` `Typecast` function to convert it to a class reference.

Using the TestStand ActiveX API in Different Programming Languages

LabVIEWAccessing Property Objects

LabWindows/CVIAccessing Property Objects

C++ using MFC.

Accessing Property Objects

Using Strings

LabVIEW accesses the TestStand ActiveX API using the invoke and property nodes in LabVIEW version 5.0 and later. If you want to use a sequence context or other object in a LabVIEW VI, wire the ActiveX reference for the object to an invoke or property node, and select

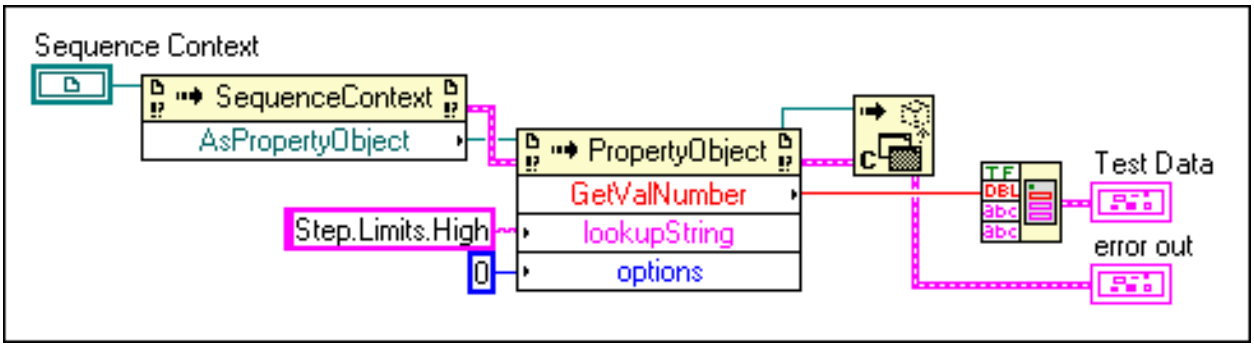
For detailed information about the LabVIEW ActiveX Automation client functions, refer to Chapter 51, *ActiveX Automation Functions* the *LabVIEW Function and VI Reference Manual*

When a method or property returns an ActiveX reference to you, you must discard it when you are done with it. Discard it by using the ActiveX Close Node. If you fail to discard it, LabVIEW discards it when your VI hierarchy finishes executing. See the topic for more information on references to ActiveX objects.

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

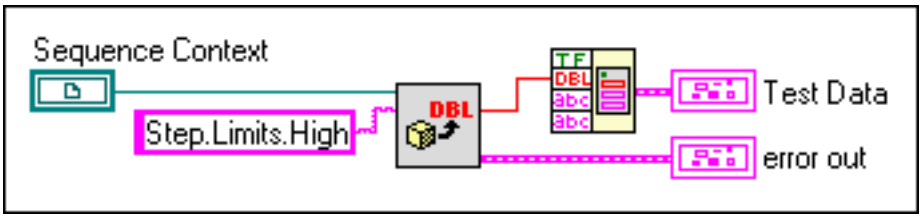
Accessing Property Objects in LabVIEW

The following test VI demonstrates how to access step properties from a sequence context parameter using the `PropertyObject` class. The VI obtains the value of the upper limit in the current step and sets the measurement value in the TestData cluster to the upper limit.



You can simplify this by using one of the VIs that comes with TestStand to access values from objects. The VIs appear in the TestStand

in the previous example.



LabWindows/CVI

LabWindows/CVI accesses the TestStand ActiveX API using the instrument driver in the `TestStand\API\CVI`

The instrument driver functions you use to invoke methods have a special naming convention. The function names start with the `TS_`

function name within the constraints of the `.fp`

You can access the values of built-in properties using *get* *set* functions in the TestStand API instrument driver. Each get or set `TS_` prefix, followed by the class name, followed by or `GetProperty` the value of a built-in property of a step, you use the `TS_StepGetProperty` function to specify a particular property.

The LabWindows/CVI ActiveX Automation Library uses the CAObjHandle data type for handles to ActiveX objects. The TestStand

for you to pass the handle for one TestStand API class to a method of a different TestStand API class. This is useful when you want to pass the sequence context handle to a class method in a test module. It is also useful when you want to pass a PropertyObject handle to a method of another class. For example, if you use the method on the sequence context to obtain a PropertyObject handle for the currently executing sequence, you can pass the handle directly to any class method. A drawback of using the same data type for all TestStand objects is that the compiler cannot flag calls to methods in which you pass a handle for the wrong kind of object.

`GetProperty` function, you must release the handle when you `CA_DiscardObjHandle` function in the LabWindows/CVI ActiveX `CA_DiscardObjHandle`, see the _____ topic.

Some TestStand ActiveX API methods have output parameters that return strings. You must free these strings when you are done with `CA_FreeMemory` function in the LabWindows/CVI ActiveX Automation Library to free these items.

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

The following test function demonstrates how to access step properties from a sequence context parameter using the `PropertyObject` tTestData structure to the upper limit.

```
void TX_TEST DLLEXPORT TestModule (tTestData *testData, tTestError *testError)

    ERRORINFO errorInfo;
    double highLimit;


"Step.Limits.High", 0, &highLimit);
    testData->measurement = highLimit;
```

The errorInfo variable is a structure that the LabWindows/CVI ActiveX Automation Library defines to hold information about errors that might occur in the operation of the function.

In Visual Basic, you can access the TestStand ActiveX API by adding the TestStand API to the project as a component or as a reference. All the classes, methods, and properties are available in Visual Basic's object browser and are accessible from your Visual Basic source

interface in Visual Basic, add the TestStand ActiveX API as a component.

MFC/C++

`tsapimfc.cpp` and files define. The `tsapimfc.cpp` `tsapimfc.h` files are in the directory. They define a set of wrapper classes that are analogous to the TestStand classes such as , `Step` `Report`.

get and function for each built-in property. The methods and property functions use the normal C++ data types for parameters and returns values. In two cases, however, you must do extra work to handle the parameters and return values correctly. object references strings.

Object reference parameters in MFC are always of type `LPDISPATCH` a wrapper class, you pass the dispatch pointer member variable of the object. For instance, to pass a sequence object to the `SequenceFile`

```
seqFile.InsertSequence(sequence.m_lpDispatch);
```

When a TestStand MFC wrapper class method returns an object pointer as a return value or as an output parameter, it returns an value. You must attach the dispatch pointer to an instance of the MFC wrapper class that corresponds to the TestStand class of the object. Each class has an method that assigns a dispatch pointer to the member variable, `m_lpDispatch,` TestStand MFC wrapper class.

```
SequenceFile mySeqFile;

dispatchPtr = engine.GetSequenceFile("c:\\seqs\\testset1.seq"));
mySeqFile.AttachDispatch(dispatchPtr);
```

attach the dispatch pointer using the `AttachDispatch` `FALSE` for the optional auto-release parameter, as

```
int MyTestFunction (LPDISPATCH sequenceContextDisp)
{

      context.AttachDispatch(sequenceContextDisp, FALSE);
}
```

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

This prevents the MFC object from incorrectly releasing the dispatch pointer when the destructor executes. Never release objects that you receive as parameters unless you first add a reference to them. See the Advanced Issues topic for more information on references to ActiveX objects.

Accessing Property Objects in MFC/C++

You can pass the dispatch pointer for an object of any TestStand class other than the `Engine` class to the `AttachDispatch` method for the `PropertyObject` class. This is useful when you want to use `PropertyObject` class methods on the sequence context in a test module. The following code shows how to use to sequence context to get the value of a custom step property for the current step.

```
int MyTestFunction (LPDISPATCH sequenceContextDisp)
{
    PropertyObject property;
    double lowLimits;

    property.AttachDispatch(sequenceContextDisp, FALSE);
    lowLimits = property.GetValNumber("Step.Limits.Low", 0);
}
```

You can also pass the dispatch pointer for an object of the `PropertyObject` class to the `AttachDispatch` method for any other TestStand class. This is useful when you want to get an object using the `GetValObject` method of the `PropertyObject` class and then call a method of another class on it. `GetValObject` returns a dispatch pointer, and you can pass that dispatch pointer to the `AttachDispatch` method for another class. The following code shows how to use the `GetValObject` method of the `PropertyObject` class to obtain an object for the currently executing sequence.

```
int MyTestFunction (LPDISPATCH sequenceContextDisp)
{
    PropertyObject property;
    Sequence sequence;

    property.AttachDispatch(sequenceContextDisp, FALSE);
    sequence.AttachDispatch(property.GetPropertyObject("RunState.Sequence", 0);
}
```

Strings in MFC/C++

Normally, the TestStand MFC class methods automatically convert string return values to the MFC `CString` type. An exception is string output parameters. They are returned as type BSTR. The following example shows how to convert a BSTR value to a `CString`. You must free BSTR strings when you are done with them.

```
CString stringVar;

stringVar = stringVarAsBStr;      /* convert BSTR to CString*/
SysFreeString(stringVarAsBStr);   /* dispose BSTR */
```

All classes in the TestStand MFC interface derive from the MFC `COleDispatchDriver` class. For more information on the `COleDispatchDriver` class, refer to the MFC documentation you received with your compiler.

Advanced Issues

Adding and Releasing References to ActiveX Objects

- LabVIEW
- LabWindows/CVI
- Visual Basic
- MFC/C++

Thread concurrency models

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

Adding and Releasing References

Whenever you create an ActiveX object directly or receive an ActiveX object as a return value from a method or property, you have a reference to the object. You must release this reference when you no longer require the object.

Normally, when you receive an object as a parameter to a function or LabVIEW VI, you do not have to add or release a reference to the object. This is because the caller of the function already holds a reference to the object. If, however, the function or VI stores the object for later use after the function or VI returns, you must add a reference to the object in the function or VI. You must release the reference when you no longer require the object.

In general, maintaining a reference to an object guarantees its existence for at least at long as you hold that reference. The object frees its resources only when you release the final reference to the object.

This topic illustrates how to add and release references to objects from each language.

LabVIEW

No cases currently exist in LabVIEW where you have to add a reference to an ActiveX object.

To release a reference to an ActiveX object in LabVIEW, wire the reference to the ActiveX Automation Close function.

LabWindows/CVI

To add a reference to an object in LabWindows/CVI, you must use functions from the ActiveX Automation Library as follows.

```
LPDISPATCH tmpDispatchPtr;
CAObjHandle newObjReferenceHandle;

CA_GetDispatchFromObjHandle(objHandle, &tmpDispatchPtr);
CA_CreateObjHandleFromIDispatch(tmpDispatchPtr, 1, &newObjReferenceHandle);
```

To release an object reference in LabWindows/CVI, you must use the following function from the ActiveX Automation Library.

```
CA_DiscardObjHandle(newObjReferenceHandle);
```

Visual Basic

Visual Basic automatically adds object references when you assign the object to another variable as in the following code:

```
Set newObjReference = objReference
```

Visual Basic also automatically releases object references when a variable or list that contains a reference to an object goes out of scope. Local variable scope ends when the function containing it returns. Static and global variable scope ends when you unload the module, form, or class object that contains the variable. You can force the release of a reference to an object by setting the variable to the value `Nothing`, as in the following code:

```
Set objReference = Nothing
```

MFC/C++

By default, if you attach a dispatch pointer to a TestStand MFC wrapper class object, the class releases a reference to the dispatch pointer when it destroys the object. This behavior, however, is not always what you want. For example, when you receive a dispatch pointer as a parameter, you must not release the reference to it. Releasing the reference can cause a fatal application error. To get around this, the `AttachDispatch` methods of the TestStand MFC wrapper classes has an optional second parameter to specify whether or not to auto-release the dispatch pointer. Pass FALSE to prevent the automatic release, as shown below:

```
int MyTestFunction (LPDISPATCH sequenceContextDisp)
{
    SequenceContext context;

    context.AttachDispatch(sequenceContextDisp, FALSE);
}
```

To add a reference to a dispatch pointer that a TestStand object uses, do the following:

```
object.m_lpDispatch->AddRef();
```

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

To explicitly release a reference to a dispatch pointer that a TestStand object uses, call the `ReleaseDispatch` method of the particular class.

All classes in the TestStand MFC interface derive from the MFC `COleDispatchDriver` class. For more information on the `COleDispatchDriver` class, refer to the MFC documentation for your compiler.

Thread Concurrency Models

In ActiveX, there are two main concurrency models for which you can initialize a thread, apartment-threaded and multi-threaded. Before you use a thread with ActiveX, you must initialize it to one of these models. In most cases, the software development environment you are using initializes the threads in your application for you.

When using the TestStand API to implement an Operator-Interface or other application program, you may use threads of either concurrency model.

When using ActiveX from within a test module, you do not have to initialize the thread's concurrency model because TestStand initializes all execution threads as multi-threaded for you.

Writing an Application with the TestStand API

Creating the TestStand Engine

UIMessages (User Interface Messages)
- Setting Up Your UIMessage Handler
- Handling Specific UIMessages
  - Execution Events
    - Start/End Events
    - Break Events
    - Trace Event
    - Resume Events
    - Execution State Change Events
    - Progress Events
  - Shutdown Events
  - User-Defined Events
  - Handling Unknown Events
- Posting UIMessages

Sequence Files
- Loading and Caching
- Structure of a Sequence File
- Adding New Steps and Sequences

Process Model Entry Points
- Getting the Model Sequence File
- Getting the Applicable Entry Point Sequences
- Creating the Menu Items
- Executing an Entry Point

Tools Menu
- Constructing the Tools Menu
- Updating the Tools Menu Item Names and Enabled States
- Executing a Tools Menu Item

Executing a Sequence

Starting an Interactive Execution
- Creating the InteractiveArgs Object
- Starting as a New Execution

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

Starting From a Suspended Execution

Login/Logout Execution

ArgumentList Parameter

Special Handling of the UIMessage UIMsg_EndExecution

Shutting Down the Engine

First Pass

Second Pass

Enforcing User Privileges

Resource Strings and Internationalization

Creating the TestStand Engine

When writing an application using the TestStand API, you must create a TestStand Engine object. Because you can create the TestStand Engine as either an ActiveX control or as an ActiveX automation server, you can use it from a variety of software development environments. If you use a development environment that supports ActiveX controls with events, such as Microsoft Visual Basic or Microsoft Visual C/C++, create the TestStand Engine object by adding the TestStand Engine control to the main form of your application. If your development environment does not support ActiveX controls, you can create the TestStand Engine object as an in-process automation server.

UIMessages

Setting Up Your UIMessage Handler

Handling Specific UIMessages

- Execution Events
  - Start/End Events
  - Break Events
  - Trace Event
  - Resume Events
  - Execution State Change Events
  - Progress Events
- Shutdown Events
- User-Defined Events
- Handling Unknown Events

Posting UIMessages

When you write an application that runs TestStand sequence files, you do not handle the details of executing each step. Instead, you call the TestStand engine to begin executing the sequence file in a new thread, and the engine notifies you asynchronously of the state of the execution. The engine notifies you of the execution state by sending you messages such as `Start`, `End`, `Break`, and `Trace`. You can update your application's user interface in response to these messages.

You can create an execution either directly or indirectly. For example, you create an execution indirectly when you load a sequence file that has a load callback. Before you create an execution directly or indirectly, you must set up a UIMessage handler. UIMessages are the way in which the engine communicates the execution states and other asynchronous information to your application. Depending on how you create the Engine object, you can set up the handler in one of several ways.

Setting Up Your UIMessage Handler

If you create the Engine as an ActiveX control, the easiest way to handle UIMessages is by implementing the UIMessageEvent callback for the ActiveX control. Whenever the Engine has a UIMessage for your application, the Engine calls the callback with a UIMessage object as a parameter.

If you do not create the Engine as an ActiveX control, you can handle UIMessages in either of the following two ways:

- When your software development environment allows you to write a function that is callable from C, you can instruct the TestStand Engine to call such a function by passing its address to the Engine RegisterUIMessageCallback method. The application thread that calls this method must regularly process Windows messages for this approach to work.

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

- When your development environment cannot create functions that are callable from C, you must establish a polling loop for UIMessages.

To establish a polling loop for UIMessages, you must first inform the Engine that you are polling. Do this by setting the Engine UIMessagePollingEnabled property to True. Then implement a polling loop by periodically checking the Engine IsUIMessageQueueEmpty property. If the queue is empty, call the GetUIMessage method to get the next UIMessage from the Engine. You must not check the queue again until you complete whatever action the UIMessage requires and release all your references to the UIMessage object. One way to implement such a loop in your application is to setup a timer callback that polls each time it executes.

Handling Specific UIMessages

The first thing to do when you receive a UIMessage is to check its Event property, the value of which is a UIMessageCode. Based on the UIMessageCode, you can determine which items, if any, to update on the user interface.

Execution Events

Most of the UIMessages that TestStand posts notify you to update an execution display. When you receive one of these UIMessages, use its Execution property to get a reference to the Execution object to which the message applies. Use the Id property of the Execution object to determine which user interface display to update. The following describes the procedures for handling the UIMessages that apply to executions:

Start/End Events – If you receive a UIMsg_StartExecution event, either create a new display for the execution, or reinitialize the display for the execution if one already exists. The execution display already exists when the user restarts the execution. If you receive a UIMsg_EndExecution event, modify your display for the execution to show that the execution has ended, release any handles to Thread, Step, or SequenceContext objects that you have for the execution, and display the execution's report if one exists.

TestStand posts the UIMsg_StartInteractiveExecution and EndInteractiveExecution events in order to notify you when the user runs steps in an execution interactively. The user can run steps interactively either in a new execution, or from an existing execution while at a breakpoint.

TestStand posts the UIMsg_StartFileExecution and UIMsg_EndFileExecution events to notify the user interface of the period during which an execution uses a particular sequence file. You can use this information to disable the editing of the sequence file or to indicate that it is in use.

Break Events – TestStand sends one of three different UIMessages when an execution enters a suspended state: UIMsg_BreakOnBreakpoint, UIMsg_BreakOnUserRequest, and UIMsg_BreakOnRunTimeError. Each of these messages indicates a different reason for the change in state.

For each of these events you might want to display several details, such as the following:

- Information about the steps in the currently executing step group
- A call stack list that identifies the locations of the active sequence calls in the current thread
- A list of the threads that the execution contains

Note: ***Currently, TestStand supports only one thread per execution.***

To display information about the steps in the currently executing step group, you must first get the foreground Thread object. To do so, get the value of the Execution ForegroundThreadIndex property, and then call the Execution GetThread method with that index. From the Thread object, you can then get the SequenceContext object that represents the top of the call stack. To do so, call the GetSequenceContext method on the Thread object. Pass 0 for the call stack index and pass a variable in which to store the frame ID. The frame ID is a unique identifier for the call stack entry and is not required in order to handle this event. It is useful when handling trace events as described later in this topic. Once you have the SequenceContext, you have access to all the information about the sequence that is currently executing. When you get the Sequence object from the SequenceContext's Sequence property, you are getting the run-time copy of the sequence. Access this runtime copy to get all the information about the steps that you require. You can also use the numerous other SequenceContext properties to get information such as the currently executing step group, the sequence file that contains the executing sequence, and other information. Refer to the SequenceContext API help for more information.

To display a call stack list, first get the number of call stack entries by getting the CallStackSize property of the current thread. Then get the SequenceContext object for each entry by passing its zero-based index to the Thread GetSequenceContext method. Use the SequenceContext CallStackName property to get the display name for an entry.

To display a thread list, first get the number of threads from the Execution NumThreads property. Then get each thread by passing its zero-based index to the GetThread method. Get the display name of the thread from its DisplayName property.

In response to the UIMsg_BreakOnRunTimeError event, display the run-time error to the user after updating the user interface. Get the run-time error message from the RunTimeErrorMessage property for the top-level SequenceContext. Resuming from a UIMsg_BreakOnRunTimeError event causes the execution to go to cleanup. To ignore the run-time error you must call the ClearCurrentRTE method of the foreground thread first. A run-time error dialog box is available in the Engine through the DisplayRunTimeErrorDialog method, or you can write your own.

Trace Event – The update to the user interface that a UIMsg_Trace event requires is very similar to that of a break event. One difference is that you use only the name of the foreground thread and the top entry in the call stack list, instead of the names of all the threads and

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

call stack entries. This is because the user interface should disallow switching threads or browsing the call stack while tracing. Also, the trace event does not indicate a suspended state, and thus you do not have to call the Execution Resume method to continue. Instead, execution continues automatically when you finish handling the UIMessage for the trace event, that is, when you release the UIMessage or you return from your event handling callback. In addition, the frame ID output parameter of the GetSequenceContext method is useful for optimizing your update of the display.

As an optimization, the user interface should hold the step group and frameId from the last trace event. If the next event is a trace event and its sequence context has the same frameId and step group, you need to update only the display of the step which was previously executed and the step that is next to be executed. Because this is typically the case when users trace through a long list of steps in the same sequence and step group, this optimization can significantly increase your application's maximum tracing speed.

If your application polls for UIMessages, the maximum tracing rate is limited by the rate at which you poll.

Resume Events – TestStand posts the UIMsg_ResumeFromBreak event to the user interface whenever an execution is about to resume from a suspended state. This event allows the user interface to update the enabled state of controls on its execution display and to show that the execution is running.

Execution State Change Events – The purpose of the execution state change events is to notify the user interface that the state of an execution is about to change. The action that these events imply does not occur until you finish handling the UIMessage, that is, when you release the UIMessage or return from your event handling callback. Notice that break events signify an execution state change as well. For more information on break events, refer to the Break Events section earlier in this topic.

The meaning of most of the execution state change events is inherent in their names. The names of the events are as follows:

UIMsg_TerminatingExecution
UIMsg_TerminationCancelled
UIMsg_TerminatingInteractiveExecution
UIMsg_AbortingExecution
UIMsg_KillingExecutionThreads

Update the execution display to inform the user of the change.

Progress Events – A step posts progress events to notify the user interface to update its progress indicator or text message if one exists. If you receive a UIMsg_ProgressPercent event, check the NumericData property of the UIMessage for the percentage of the progress indicator to shade. If you receive a UIMsg_ProgressText event, you can find the text to display in the StringData property. For more information on posting progress events, refer to the Posting UIMessages section later in this topic.

Shutdown Events

TestStand sends either the UIMsg_ShutDownComplete or the UIMsg_ShutDownCancelled event at some point after you call the Engine.ShutDown method. The event informs you that your shutdown request has ended. Refer to *Shutting Down the Engine* for more information on the shutdown procedure.

User-Defined Events

You can post your own messages with your own user-defined events by using an event number that is greater than or equal to the value of the UIMsg_UserMessageBase constant. For more information on posting events, refer to the Posting UIMessages section later in this topic.

Handling Unknown Events

If you receive a UIMessage with an event ID that your program does not expect, perform one of the following actions:

- If you are polling, release the UIMessage and continue polling.
- If you receive the UIMessage in a callback, simply return from your callback without responding to the message.

Do not treat UIMessages with unexpected event IDs as errors. Ignoring unknown events ensures that your application works with future versions of TestStand and behaves consistently with unknown user-defined events.

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

Posting UIMessages

Currently, TestStand supports posting UIMessages by calling the Thread.PostUIMessage method. This allows the steps of an execution to post progress events and user-defined events. Because this function is a method of the Thread class, the thread and its associated execution are automatically properties of the UIMessage. You can send additional data using the numeric data and string data parameters of the method. For more information, refer to the Progress Events and User-Defined Events sections earlier in this topic.

Sequence Files

Loading and Caching

Structure of a Sequence File

Adding New Steps and Sequences

Loading and Caching

To get a reference to a sequence file on disk, call the Engine method GetSequenceFile. If it is not already in memory, this method loads the sequence file from disk and places it in the Engine's internal sequence file cache. If the sequence file is already in memory, the method returns a reference to the file in the cache instead. Also, every sequence file reference you obtain from GetSequenceFile causes the cache to increment an internal usage count for that sequence file. Before you release your reference to the sequence file, you must call the Engine method ReleaseSequenceFile to decrement the usage count.

To create a new sequence file, call the Engine method NewSequenceFile. This method adds the new sequence file to the Engine's internal cache. Therefore, before you release a sequence file you obtain from NewSequenceFile, you must call ReleaseSequenceFile on it. After you create a new sequence file, it contains one sequence called MainSequence. You can create and insert additional sequences or steps as described in the *Adding New Steps and Sequences* section later in this topic.

Structure of a Sequence File

A sequence file contains an array of sequences. Each sequence contains three arrays of steps: setup steps, main steps, and cleanup steps. Each array of steps is called a step group. To access the sequences in a sequence file, use the following methods and properties of the SequenceFile object: NumSequences, GetSequence, GetSequenceByName, and GetSequenceIndex. To access the steps in a sequence, use the GetNumSteps, GetStep, GetStepByName, and GetStepIndex methods of the Sequence object. Notice that the all functions for accessing steps take a StepGroups constant to specify the step group that contains the step.

Adding New Steps and Sequences

Every step that you insert into a sequence must be a unique instance. If you want to use the same step in two sequences, you must insert separate copies of the step. If you want to repeat a step in a sequence, you must insert a separate copy of the step. You can make a copy of a step by calling the PropertyObject Clone method. Thus, you can insert a step after you create it, clone it, or remove it from a sequence. The same applies to inserting sequences into sequence files.

To create a new step, call the Engine method NewStep, and pass an adapter key name and a step type name. Use the AdapterKeyNames constants and the StepTypes constants. You can then insert the new step into a sequence using the Sequence InsertStep method. Be sure to name the new step, and set its adapter and step-type-specific properties as necessary.

To create a new sequence, call the Engine method NewSequence and insert the new sequence into a sequence file using the SequenceFile method InsertSequence. Be sure to name the new sequence. Use a name that is unique among the sequences in the sequence file.

Other methods you use to implement sequence file editing are RemoveSequence, DeleteSequence, SequenceNameExists, RemoveStep, DeleteStep, and StepNameExists.

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

Process Model Entry Points

Getting the Model Sequence File

Getting the Applicable Entry Point Sequences

Creating the Menu Items

Executing an Entry Point

Before reading this topic, be sure to read the *Process Models* subsection of the *TestStand Building Blocks* section of Chapter 1, *TestStand Architecture Overview*, in the *TestStand User Manual*. This topic requires a thorough understanding of process models and how you use them in TestStand.

Getting the Model Sequence File

Applications typically display the entry points that the process model defines as menu items or buttons that the user can select. To determine the process model entry points that apply to a given situation, you must first get the process model sequence file. To get the model sequence file, do the following:

1) If you are displaying an execution, call the GetSequenceFile method of the execution and call GetModelSequenceFile on the sequence file that GetSequenceFile returns. If GetModelSequenceFile returns a NULL reference, go to step 3. Otherwise, you have the reference to the model sequence file. The sequence file that GetSequenceFile returns is the client sequence file.

2) If you are displaying a sequence file, call the GetModelSequenceFile method on the sequence file. If GetModelSequenceFile returns a NULL reference, go to step 3. Otherwise, you have the model sequence file. The sequence file that you are displaying is the client sequence file.

3) If you are not displaying a sequence file or an execution, or if the execution or sequence file does not have a model sequence file, call the Engine GetStationModelSequenceFile method. If GetStationModelSequenceFile returns a NULL reference, then no model sequence file currently applies. Otherwise, GetStationModelSequenceFile returns the applicable model sequence file. In this case, no client sequence file exists.

Getting the Applicable Entry Point Sequences

If no model sequence file applies to the current context in the user interface, then no entry points currently apply. If a model sequence file does apply, you can determine which, if any, sequences in it are entry points and whether they apply. To do so, enumerate through all the sequences in the model sequence file. First, get the number of sequences from the NumSequences property of the model sequence file. Then get each individual sequence using the GetSequence method. To get each sequence, pass the indices 0 through (NumSequences – 1). For each sequence, determine if it is an applicable entry point according to the following procedure:

1) Is the Type property of the sequence equal to SeqType_CfgEntryPoint? If so, go to step 5. If not, go to step 2.

2) Is the Type property of the sequence equal to SeqType_ExeEntryPoint? If so, go to step 3. If not, then this sequence is not an applicable entry point.

3) Did you obtain the model sequence file from a client sequence file? If so, go to step 4. If not, then this sequence is not an applicable entry point. Notice that the latter case applies when you obtain the model sequence file by calling GetStationModelSequenceFile,

4) Does the client sequence file from which you obtained the model sequence file have a sequence called "MainSequence"? If so, go to step 5. If not, then this sequence is not an applicable entry point.

5) If the ShowEntryPointForEditorOnly property of the sequence is True and your application is not a sequence editor, then this sequence is not an applicable entry point. If ShowEntryPointForEditorOnly is False or your application is a sequence editor, go to step 6.

6) If the ShowEntryPointForExeWindow property of the sequence is True and you are displaying an execution window, then this sequence is an applicable entry point. Add it to your list of entry point sequences. If not, go on to step 7.

7) If the ShowEntryPointForFileWindow property of the sequence is True and you are displaying a sequence file, then this sequence is an applicable entry point. Add it to your list of entry point sequences. If not, go on to step 8.

8) If the ShowEntryPointForAllWindows property of the sequence is True, then this sequence is an applicable entry point. Add it to your list of entry point sequences. If not, then this sequence is not an applicable entry point.

You now have a list of applicable entry point sequences. The next step is to create the menu items and/or command buttons that allow the user to create executions from the entry points.

Creating the Menu Items

When creating an entry point, you must determine which menu or submenu to place it in. You can place the entry point by type, putting entry points of type SeqType_CfgEntryPoint in the Configure menu and entry points of type SeqType_ExeEntryPoint in the **Execute** menu, or you can use the GetEntryPointMenuFromHint method of the entry point sequence. Refer to the help for the for

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

GetEntryPointMenuFromHint method more information. If GetEntryPointMenuFromHint fails to find a menu that matches any of the entry point's menu hints, then place the entry point menu item by type.

After you determine the location of the menu item, get the string to display for the menu item by calling the EvalEntryPointNameExpression method of the entry point sequence. Pass a reference to the client sequence file as the parameter to the method, or pass a NULL reference if no client sequence file exists. Similarly, call EvalEntryPointEnabledExpression to determine whether to enable the menu item. Besides checking the value that EvalEntryPointEnabledExpression returns, you might want to check certain user privileges such as Priv_Execute or Priv_RunAnySequence. For more information on checking user privileges, refer to *Enforcing User Privileges* in this help document.

Executing an Entry Point

This section discusses how to execute an entry point. If a client sequence file exists, you must first create an EditArgs object by calling the Engine method NewEditArgs. You use the EditArgs object to specify which TestStand objects the user selects in your application. If the client sequence file is from an execution, call the SetSelectedExecution method of the newly created EditArgs object, and pass a reference to the execution. If the client sequence file is from a sequence display, then call SetSelectedSequenceFile instead, and call AddSelectedSequence and AddSelectedStep to add the currently selected steps and sequences, if any.

Pass the following as arguments to the Engine's NewExecution method:

Parameter Name:	**What to Pass:**
SequenceFileParam	A reference to the client sequence file, or a NULL reference if there is no client sequence file. Depending on the development environment you use, pass 0 or the Nothing keyword to indicate a NULL reference.
SequenceName	The value of the Name property of the entry point sequence.
ProcessModelParam	A reference to the model sequence file
BreakAtFirstStep	True or False
ExecutionTypeMaskParam	If the value of the EntryPointInitiallyHidden property of the entry point sequence is True, then pass the value of the ExecTypeMask_InitiallyHidden constant. Otherwise, pass ExecTypeMask_Normal.
SequenceArgsParam (optional)	Leave this optional parameter unspecified.
EditArgsParam (optional)	You have to specify this parameter only if a client sequence file exists. Pass a reference to the EditArgs object that you create according to the instructions earlier in this section.
InteractiveArgsParam (optional)	You have to specify this parameter only if you want to create an interactive execution. For more information on creating interactive executions, refer to *Starting an Interactive Execution*.

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

Tools Menu

Constructing the Tools Menu
Updating the Tools Menu Item Names and Enabled States
Executing a Tools Menu Item

This topic explains how to implement the TestStand **Tools** menu in your application.

Constructing the Tools Menu

To implement a **Tools** menu, you must first call the Engine ConstructToolMenus method. This method returns the number of tool menus, which includes the top level menu and all of its submenus. The ConstructToolMenus method and the EvalToolMenuItemExprs method that you use later each take an EditArgs object as a parameter. You pass an EditArgs object to indicate which execution, sequences, and steps are selected when the user chooses a **Tools** menu item. The name expressions and enabled expressions of a **Tools** menu item can use this information to determine the name or enabled state. Also, if the **Tools** menu item executes a sequence, the information you provide in the EditArgs object is available to the execution. **Tools** menu items can use the selection information in the EditArgs object to act upon the selected sequence or step.

There are two approaches to obtaining the menu items for the **Tools** menu and its submenus. One approach is to start from menu index 0, which represents the top-level menu, and get the menu item information for the submenus as you encounter the menu items that own the submenus. The alternative is to get all the menu items of all the menus first and then link the menus by identifying each menu item that owns a submenu. You can get the number of menu items in each menu by passing the menu indices to the GetNumToolMenuItems method. The menu indices are 0 through (Number of Menus – 1). The menu index 0 always indicates the top-level **Tools** menu. The rest of the menus are submenus. For each menu item, you can call GetToolMenuItemInfo to get the name, the enabled state, a unique ID that you can use to refer to the menu item later, and the index of the submenu that the item owns. If the item does not own a submenu, TestStand returns a submenu index of -1. If it owns a submenu, the submenu index is the menu index of the submenu, and you can use it with GetNumToolMenuItems, GetToolMenuItemInfo, and all other methods that take a menu index parameter.

In addition to displaying the **Tools** menu items in your application, you might want to add a **Customize** menu item to allow the user to configure the **Tools** menu. To do so, add a menu item to the **Tools** menu that calls the Engine DisplayToolMenuDialog method.

Updating the Tools Menu Item Names and Enabled States

When the user selects a different execution, set of sequences, or set of steps, or when the user pulls down a menu, call the EvalToolMenuItemExprs method to update the names and enabled states of the menu items. Then get the updated menu item information for all the menu items by using each menu item's unique ID with the GetToolMenuItemInfoWithID method, or by calling GetToolMenuItemInfo.

Note: ***If the user alters a* Tools *menu item through the dialog box that appears when you call* DisplayToolMenuDialog*, you must reconstruct the* Tools *menu. Follow the instructions earlier in this topic for constructing the* Tools *menu.***

Executing a Tools Menu Item

To execute a **Tools** menu item, call the Engine InvokeToolMenuItemWithID method with the menu item ID, or call InvokeToolMenuItem with the index of the menu and the index of the menu item. Executing a **Tools** menu item can do several types of things, such as running a sequence or launching an executable.

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

Executing a Sequence

The most direct way to create an execution in TestStand is to execute a specific sequence in a specific sequence file. Although doing so bypasses the process model, it is useful to support such a feature in your application for debugging purposes and to give the user additional control. To execute a particular sequence, call the Engine method NewExecution passing the following arguments:

Parameter Name:	**What to Pass:**
SequenceFileParam	A reference to the sequence file that contains the sequence you want to execute.
SequenceName	A string containing the name of the sequence to execute.
ProcessModelParam	A NULL reference. Depending on the development environment you use, pass 0 or the Nothing keyword to indicate a NULL reference.
BreakAtFirstStep	True or False.
ExecutionTypeMaskParam	The value of the constant ExecTypeMask_Normal.
SequenceArgsParam (optional)	Leave this optional parameter unspecified.
EditArgsParam (optional)	Leave this optional parameter unspecified.
InteractiveArgsParam (optional)	Leave this optional parameter unspecified.

Starting an Interactive Execution

Creating the InteractiveArgs Object

Starting as a New Execution

Starting From a Suspended Execution

There are two different ways to start an interactive execution in TestStand. You can start one from an execution that is suspended at a breakpoint using the Thread DoInteractiveExecution method, or you can start one as a new execution using the Engine NewExecution method.

Creating the InteractiveArgs Object

When you start an interactive execution, you create an InteractiveArgs object and then pass it to DoInteractiveExecution or NewExecution. You use the InteractiveArgs object to specify which steps to run and whether and how the execution loops.

To determine whether and how the user of your application wants the execution to loop, use the dialog box that TestStand provides through the DisplayLoopOnStepsDialog method, or write your own dialog box. Next, call the Engine NewInteractiveArgs method to create the InteractiveArgs object. Set the StepGroup, LoopCount, and StopExpression properties of the object appropriately. You do not have to set the LoopCount or StopExpression if you do not want the interactive execution to loop. Finally, call the InteractiveArgs method AddStepIndex for each step that you want to run, passing the zero-based step index for each step. In an interactive execution, you can run only steps that are together in the same sequence and step group.

You can then pass your InteractiveArgs object to NewExecution or DoInteractiveExecution. Remember to release your reference to the InteractiveArgs object after you start the execution.

Starting as a New Execution

Creating a new interactive execution is the same as creating a new execution to execute a specific sequence, except that you pass an InteractiveArgs parameter. Create an InteractiveArgs object, and pass it to the NewExecution function. For more information on creating an execution to execute a specific sequence, refer to *Executing a Sequence*.

Starting From a Suspended Execution

If you are currently displaying an execution that is at a breakpoint, you can start an interactive execution on steps in the current step group and in the current thread of the execution. To do so, call the DoInteractiveExecution method on the current thread with an InteractiveArgs object.

Note: ***You cannot start an interactive execution from a breakpoint in another interactive execution.***

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

Login/Logout Execution

ArgumentList Parameter

Special Handling of the UIMessage UIMsg_EndExecution

In TestStand, the login and logout procedure is implemented as a front-end callback. This means that the code that prompts the user for a login and processes it is in a sequence that you call from your application. Call the LoginLogout front-end callback when your application starts up, when you want to give the user the ability to login or logout, and when you want the user to logout before you shut down the Engine. You call the LoginLogout callback by using the Engine CallFrontEndCallback method and passing "LoginLogout" for the sequence name parameter.

The LoginLogout callback logs out the currently logged-in user. Depending on the value of the arguments in the argumentList parameter, it can also log in a new user.

ArgumentList Parameter

The argumentList parameter to CallFrontEndCallback is a property object that contains a set of subproperties. Each subproperty represents an argument. To create the PropertyObject object for the argumentList, call the Engine NewPropertyObject method as follows:

```
NewPropertyObject (PropValType_Container, False, "", 0)
```

Call the SetValBoolean method on the newly created PropertyObject to set the value of the first argument, logoutOnly. You can choose a true or a false setting as shown in the following samples:

- `SetValBoolean ("logoutOnly", PropOption_InsertIfMissing, True)`
- `SetValBoolean ("logoutOnly", PropOption_InsertIfMissing, False)`

Finally, call the SetValBoolean method to set the value of the second argument, isInitialLogin. This argument indicates whether this is the initial login at application startup. The callback uses this information to implement the Automatically Login Windows System User feature.

After you call CallFrontEndCallback, release the PropertyObject that you created for the argumentList parameter.

Special Handling of the UIMessage UIMsg_EndExecution

To determine when a logout or login/logout operation has completed, keep track of the Id of the execution that CallFrontEndCallback returns. When you receive the UIMsg_EndExecution UIMessage, you can use this handle to determine if the execution that completed is the front-end callback that you created. Knowing when the login/logout callback completes allows you to do the following:

1) Update the display to show the name of the user currently logged-in. To do this, get the User object for the current user from the Engine CurrentUser property, and get the name of that user from the User LoginName property. You can use the FullName property if you prefer.

2) Process command line arguments after a user has logged in. It is a good idea to refrain from loading a sequence file or running an execution in response to command line arguments until you give the user a chance to log in.

3) Shutdown after the last user logs out. Refer to *Shutting Down the Engine* for more information.

4) Update the state of menu items and other controls that depend on user privileges. You must reevaluate menu items and other controls that relate to model entry points. The results of the name expressions and enabled expressions for entry points can vary based on the current user.

Shutting Down the Engine

First Pass

Second Pass

There are two situations in which you perform a shutdown of the Engine. One is just before you exit your application, and the other is when you log out the current user. Shutting down the engine is a two-pass process that allows you to release sequence files and wait for their unload callbacks to complete.

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

First Pass

1) Unload all sequence files and close all completed executions. Your application must release all references to sequence files and executions so that TestStand can unload all associated sequence files and run any unload callbacks that they might have.

2) Call the Engine's ShutDown method passing False for the final parameter

The first pass ends when you receive the UIMsg_ShutDownComplete or UIMsg_ShutDownCancelled UIMessage. If you receive the UIMsg_ShutDownComplete event, continue with pass two. If you receive the UIMsg_ShutDownCancelled event instead, then resume the normal operation of your application. The user can cancel the shutdown by pressing the **Cancel** button in the Waiting for Execution to Complete timeout dialog box that TestStand displays for executions that do not end within their allotted time. For more information on UIMessages refer to *UIMessages*.

Second Pass

1) Close all executions that TestStand created as a result of the first pass. Release references to all executions and close the display windows that you used to display them.

2) Call the LoginLogout front-end callback passing True for the logoutOnly argument. Refer to *Login/Logout Execution* for more information.

3) When the LoginLogout callback completes and you receive the UIMsg_EndExecution event, call the Engine ShutDown method again, this time passing True for the final parameter.

The second pass ends when you receive the UIMsg_ShutDownComplete event. When you receive UIMsg_ShutDownComplete, and you are shutting down in preparation to exit your application, it is then safe for your application to exit. If you are shutting down because the current user logged out, you can resume normal operation.

Enforcing User Privileges

Most of the dialog boxes that you can display using methods in the Engine verify user privileges for you and disable controls that the user does not have permission to access. However, the remainder of the API does not enforce privileges. This is necessary to avoid overly restrictive situations. For example, if the TestStand API strictly enforced the execution privilege, then your application could not run the LoginLogout callback to allow a user to log in.

It is the responsibility of the application developer to enforce user privileges. To make this easier, TestStand provides a pre-defined set of user privileges along with an engine method that you can call to check if the current user has a specific privilege. To determine whether to disable controls or menu items in your application, call the Engine CurrentUserHasPrivilege method. Pass the value of one of the UserPrivileges string constants for the privilegeName parameter. If no user is currently logged in, then CurrentUserHasPrivilege always returns False. If privilege checking is disabled, then CurrentUserHasPrivilege always returns True.

Resource Strings and Internationalization

To provide both the application developer and the end-user a way to internationalize the text of applications and dialog boxes, TestStand provides a string file mechanism for storing all the application text in a central location. This allows you to translate the text more easily. You can change the strings that your application displays by editing or creating `.ini` files in a text editor. For more information on the file format and the resource string mechanism, refer to *Creating String Resource Files* section of Chapter 3, *Configuring and Customizing TestStand,* in the *TestStand User Manual*.

To add string support for the internationalization of your application, use the Engine method GetResourceString to get the text for your windows and controls. You can add your strings to an existing string file or create a new string file in the appropriate directory under the `TestStand\Components\User\Language` directory. Create a unique category for your strings.

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

Class Method and Property Tree

PropertyObject
- Methods
  - Values
    - GetValNumber
    - SetValNumber
    - GetValBoolean
    - SetValBoolean
    - GetValString
    - SetValString
    - GetValIDispatch
    - SetValIDispatch
    - GetValInterface
    - SetValInterface
    - GetValVariant
    - SetValVariant
  - Other Information
    - GetFlags
    - SetFlags
    - GetType
    - GetTypeDefinition
    - GetDimensions
    - SetDimensions
    - GetArrayIndex
    - GetArrayOffset
  - SubProperties
    - NewSubProperty
    - DeleteSubProperty
    - GetNumSubProperties
    - GetNthSubPropertyName
    - SetNthSubPropertyName
    - GetPropertyObject
    - SetPropertyObject
  - Miscellaneous
    - Exists
    - Clone
    - Evaluate
    - Write
    - Read
    - Serialize
    - Unserialize
- Static Properties
  - Name
  - Comment

Advanced

Engine
- Methods
  - ShutDown
  - Object Creation/Access
    - NewSequenceFile

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

GetSequenceFile
ReleaseSequenceFile
NewSequence
NewStep
NewUser
GetUser
GetUserProfile
UserNameExists
GetAdapter
NewExecution
CallFrontEndCallback
NewInteractiveArgs
NewEditArgs
NewPropertyObject
GetUIMessage
Execution Control
AbortAll
BreakAll
TerminateAll
Dialogs
DisplayLoginDialog
DisplayLoopOnStepsDialog
DisplayEditUserDialog
DisplayNewUserDialog
DisplaySearchDirDialog
DisplayOptionsDialog
DisplayRunTimeErrorDialog
DisplayOpenFileDialog
DisplayStepPropDialog
DisplaySequencePropDialog
DisplaySeqFilePropDialog
DisplayPreconditionDialog
DisplayBrowseExprDialog
DisplayToolMenuDialog
DisplayExternalViewerDialog
DisplayAdapterConfigDialog
NotifyStartOfModalDialog
NotifyEndOfModalDialog
Tool Menus
ConstructToolMenus
GetNumToolMenus
GetNumToolMenuItems
GetToolMenuItemInfo
GetToolMenuItemInfoWithID
InvokeToolMenuItem
InvokeToolMenuItemWithID
EvalToolMenuItemExprs
Files and Paths
FindFile
FindPath
CreateTempFile
Miscellaneous
CurrentUserHasPrivilege
CheckExprSyntax

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

GetErrorString
GetResourceString
LaunchExternalViewer
RegisterUIMessageCallback
CommitGlobalsToDisk
ReloadGlobals
UnloadAllModules
GetStationModelSequenceFile
Static Properties
Globals
CurrentUser
ApplicationIsEditor
AppMainHwnd
NumAdapters
DefaultAdapter
EnableUserPrivilegeChecking
AutoLoginSystemUser
Version Information
MajorVersion
MinorVersion
RevisionVersion
VersionString
Directories
ConfigDirectory
BinDirectory
TestStandDirectory
Types
StepTypes
CustomDataTypes
BuiltinDataTypes
Execution Options
TracingEnabled
BreakpointsEnabled
DisableResults
AlwaysGotoCleanupOnFailure
BreakOnRTE
ExecutionMask
UI Message
UIMessagePollingEnabled
IsUIMessageQueueEmpty
UIMessageDelay
UIMessageMinDelay
SequenceFile
Methods
GetSequence
GetSequenceByName
InsertSequence
RemoveSequence
DeleteSequence
SequenceNameExists
GetSequenceIndex
Save
GetModelSequenceFile
CreateCallbackOverrideSequence

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

NewEditContext
IncChangeCount
AsPropertyObject
Static Properties
NumSequences
FileGlobalsDefaultValues
Path
ModuleLoadOption
ModuleUnloadOption
HasModel
UnloadCallbackEnabled
IsExecuting
ChangeCount
Sequence
Methods
GetNumSteps
GetStep
GetStepByName
InsertStep
RemoveStep
DeleteStep
StepNameExists
GetStepIndex
EvalEntryPointNameExpression
EvalEntryPointEnabledExpression
GetEntryPointMenuFromHint
AsPropertyObject
Static Properties
Parameters
Locals
GotoCleanupOnFailure
DisableResults
ShowEntryPointForEditorOnly
ShowEntryPointForFileWindow
ShowEntryPointForExeWindow
ShowEntryPointForAllWindows
EntryPointInitiallyHidden
Type
Name
Step
Methods
ExecuteEditSubstep
SpecifyModule
AsPropertyObject
Static Properties
CanExecuteEditSubstep
CanSpecifyModule
IsSequenceCall
AdapterKeyName
BreakOnStep
RunMode
RunTimeRunMode
StepFailCausesSequenceFail
Precondition

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

Post Actions
PassAction
PassActionTarget
FailAction
FailActionTarget
CustomTrueAction
CustomTrueActionTarget
CustomFalseAction
CustomFalseActionTarget
CustomActionExpression
Looping
LoopType
LoopWhileExpression
LoopStatusExpression
LoopInitExpression
LoopIncExpression
StatusExpression
ResultStatus
PreExpression
PostExpression
RecordResult
ModuleLoadOption
ModuleUnloadOption
Name
Description
Icon
SmallIcon
LargeIcon
SmallIconIndex
LargeIconIndex
IconName

User

Methods
ValidatePassword
HasPrivilege
AsPropertyObject
Static Properties
LoginName
Password
FullName
Privileges

Adapter

Methods
Configure
AsPropertyObject
Static Properties
IsConfigurable
KeyName
DisplayName
Icon
SmallIcon
LargeIcon
SmallIconIndex
LargeIconIndex

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

IconName
Execution
Methods
Break
Resume
StepOver
StepInto
StepOut
Abort
Terminate
TerminateInteractiveExecution
CancelTermination
Restart
GetThread
GetSequenceFile
GetModelSequenceFile
AddExtraResult
DeleteExtraResult
ClearExtraResultList
GetStates
WaitForEnd
AsPropertyObject
Static Properties
InInteractiveMode
TypeMask
BreakOnRTEForThisExecution
Threads
NumThreads
ForegroundThreadIndex
Callbacks
SeqFilePreStepCallbackEnabled
SeqFilePostStepCallbackEnabled
ModelPreStepCallbackEnabled
ModelPostStepCallbackEnabled
StationPreStepCallbackEnabled
StationPostStepCallbackEnabled
SeqFilePreInteractiveCallbackEnabled
SeqFilePostInteractiveCallbackEnabled
ModelPreInteractiveCallbackEnabled
ModelPostInteractiveCallbackEnabled
StationPreInteractiveCallbackEnabled
StationPostInteractiveCallbackEnabled
Result Options
TimeResultsEnabled
StandardResultsEnabled
ResultStatus
ErrorObject
ResultObject
Report
Id
DisplayName
Thread
Methods
ClearTemporaryBreakpoint

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

SetStepOver
SetStepInto
SetStepOut
GetSequenceContext
ClearCurrentRTE
DoInteractiveExecution
PostUIMessage
AsPropertyObject
Static Properties
DisplayName
Execution
Id
CallStackSize
SequenceContext
Methods
IsInteractiveStep
AsPropertyObject
Static Properties
Step
PreviousStep
NextStep
Sequence
SequenceFile
StepIndex
PreviousStepIndex
NextStepIndex
SequenceIndex
StepGroup
Root
Main
Caller
CallStackName
CallStackDepth
RunTimeErrorMessage
StepGroupStartedInteractiveExe
InInteractiveMode
InteractiveContext
Thread
Execution
Report
IsProcessModel
ProcessModelClient
SelectedFile
SelectedSequences
SelectedSteps
SelectedExecution
Locals
Parameters
FileGlobals
StationGlobals
LoopIndex
LoopNumPassed
LoopNumFailed
Tracing

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

SequenceFailed
ApplicationIsEditor
Engine
InteractiveContext
Methods
AsPropertyObject
Static Properties
IsRootExecution
SavedPreviousStepIndex
SavedNextStepIndex
SavedStepIndex
InteractiveArgs
Methods
GetStepIndex
AddStepIndex
ContainsStep
ClearStepList
AsPropertyObject
Static Properties
StepGroup
NumSteps
LoopCount
StopExpression
EditArgs
Methods
SetSelectedExecution
SetSelectedSequenceFile
AddSelectedSequence
AddSelectedStep
ClearSelectedSequences
ClearSelectedSteps
AsPropertyObject
UIMessage
Methods
AsPropertyObject
Static Properties
Event
IsSynchronous
Execution
Thread
NumericData
StringData
Report
Methods
Append
Reset
Save
Load
LaunchViewer
GetTempFile
AsPropertyObject
Static Properties
All

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

Id
Location
Format

Classes, Constants, and Enumerations

Classes

Adapter
EditArgs
Engine
Execution
InteractiveArgs
InteractiveContext
PropertyObject
Report
Sequence
SequenceContext
SequenceFile
Step
Thread
UIMessage
User

Constants

AdapterKeyNames
CVIStepAdditions
DefaultModelCallbacks
ExecutionMask
ExecutionTypeMask
FlexCStepAdditions
GetSeqFileFlags
LVStepAdditions
NamedPropertyTypes
OpenFileDialogFlags
PostActionValues
PropertyFlags
PropertyOptions
ReadWriteOptions
RunModes
SeqFileCallbacks
SequenceCallStepAdditions
SequenceContextProperties
SequenceProperties
SpecifyModuleOptions
StepLoopTypes
StepProperties
StepTypes
UserPrivileges

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

Enumerations

- FindFilePromptOptions
- FindFileSearchListOptions
- FindPathStatusValues
- ModuleLoadOptions
- ModuleUnloadOptions
- PropertyValueTypes
- ReportConversion
- RTEOptions
- SequenceTypes
- StepGroups
- TSError
- UIMessageCodes
- ExecutionRunStates
- ExecutionTerminationStates

lookupString

To specify the dynamic property that a method operates on, you pass a string that defines a complete path from the object on which you call the method to the specific property you want to access. This string is called a lookupString. To specify the object itself, pass an empty string (""). To specify a subproperty, pass the name of the subproperty. To specify a subproperty of a subproperty, pass a string containing both names separated by a period(.). For example, you can specify the error code property in a step object using the following lookupString:

```
"Result.Error.Code"
```

For more information see the Using the PropertyObject Class to Access Dynamic Properties section.

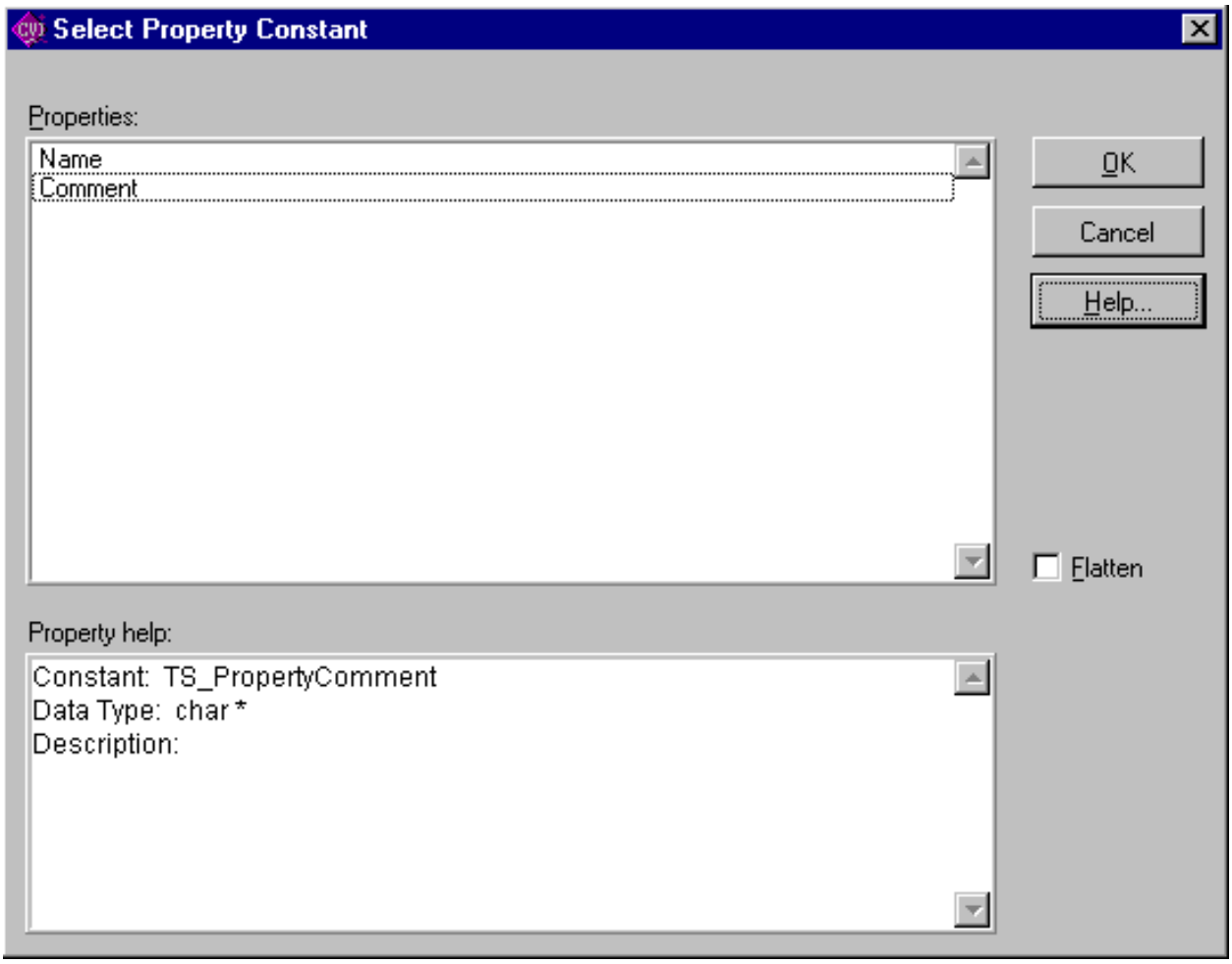
LabWindows/CVI Function Panels for Getting/Setting Static Properties

TestStand defines a number of built-in properties that are always present for objects such as steps and sequences. These properties are called static properties. Almost every class of object in TestStand has static properties. The TestStand API instrument driver for LabWindows/CVI contains functions to get and set the values of the static properties. Each class has its own get and set functions.

A LabWindows/CVI function panel is available for each get and set function. Each function panel has a Property ID control in which you specify the static property. When you click in the control, the Select Property Constant dialog box appears. The dialog box lists the static properties for the class. It is very similar to the Select Attribute Constant dialog box that you use to select attributes in the User Interface Library function panels.

To view the help for a particular property, bring up the Select Property Constant dialog box, select the property in the list box, and click the Help button. Notice that the Description field in the Property Help text box is always empty.

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.



© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

TestStand API Reference

Adapter
AdapterKeyNames
CVIStepAdditions
DefaultModelCallbacks
EditArgs
Engine
Execution
ExecutionMask
ExecutionTypeMask
FlexCStepAdditions
GetSeqFileFlags
InteractiveArgs
InteractiveContext
LVStepAdditions
NamedPropertyTypes
OpenFileDialogFlags
PostActionValues
PropertyFlags
PropertyObject
PropertyOptions
ReadWriteOptions
Report
RunModes
SeqFileCallbacks
Sequence
SequenceCallStepAdditions
SequenceContext
SequenceContextProperties
SequenceFile
SequenceProperties
SpecifyModuleOptions
Step
StepLoopTypes
StepProperties
StepTypes
Thread
UIMessage
User
UserPrivileges

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

Adapter

Use adapter objects to configure and obtain information about the module adapters that come with TestStand. Refer to Chapter 12, *Module Adapters*, of the *TestStand User Manual*, for more information about module adapters. Generally, you use this class only when writing an operator interface.

To get an adapter object, call the Engine.GetAdapter method. Pass a zero-based index to specify the adapter object for which you want to obtain a reference. To find out the number of adapters that are available, get the value of the Engine.NumAdapters property.

Properties

DisplayName
IconName
IsConfigurable
KeyName
LargeIcon
LargeIconIndex
SmallIcon
SmallIconIndex

Methods

AsPropertyObject
Configure

See Also

Engine.DefaultAdapter
Engine.NumAdapters
Engine.GetAdapter

DisplayName Property (Read Only)

Syntax

Adapter.**DisplayName**()

Data Type

String

Purpose

Returns the name to display for the adapter.

IconName Property (Read Only)

Syntax

Adapter.**IconName**()

Data Type

String

Purpose

Returns the filename of the icon for the adapter.

Remarks

Icon files are in the TestStand\Components\NI\Icons directory and the TestStand\Components\User\Icons directory.

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

IsConfigurable Property (Read Only)

Syntax
Adapter.**IsConfigurable**()

Data Type
Boolean

Purpose

Returns True if the adapter has a configuration dialog box that you can display using the Configure method of this class.

See Also
Configure

KeyName Property (Read Only)

Syntax
Adapter.**KeyName**()

Data Type
String

Purpose

Returns the name that other TestStand API functions, such as Engine.NewStep, use to refer to the adapter.

See Also
Engine.NewStep
Engine.DefaultAdapter
Step.AdapterKeyName

LargeIcon Property (Read Only)

Syntax
Adapter.**LargeIcon**()

Data Type
Picture

Purpose

Returns a large icon that represents the adapter.

See Also
LargeIconIndex
SmallIcon
SmallIconIndex

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

LargeIconIndex Property (Read Only)

Syntax
Adapter.**LargeIconIndex**()

Data Type
Long

Purpose

Returns a unique index for the large icon that represents the adapter.

See Also
LargeIcon
SmallIcon
SmallIconIndex

SmallIcon Property (Read Only)

Syntax
Adapter.**SmallIcon**()

Data Type
Picture

Purpose

Returns a small icon that represents the adapter.

See Also
LargeIcon
LargeIconIndex
SmallIconIndex

SmallIconIndex Property (Read Only)

Syntax
Adapter.**SmallIconIndex**()

Data Type
Long

Purpose

Returns a unique index for the small icon that represents the adapter.

See Also
LargeIcon
LargeIconIndex
SmallIcon

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

AsPropertyObject Method

Syntax
Adapter.**AsPropertyObject**()

Return Type
PropertyObject

Purpose

Returns the underlying PropertyObject that represents the adapter. Use the PropertyObject to edit, add, or remove custom properties of the Adapter.

Remarks

Typically, Adapters do not have custom properties.

Configure Method

Syntax
Adapter.**Configure**()

Purpose

Displays the configuration dialog for the adapter. When IsConfigurable is False, a call to this method has no effect.

See Also
IsConfigurable

AdapterKeyNames Constants

Use these names when calling TestStand API functions, such as Engine.NewStep.

- AutomationAdapterKeyName
- FlexCAdapterKeyName
- GAdapterKeyName
- NoneAdapterKeyName
- SequenceAdapterKeyName
- StdCVIAdapterKeyName

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

CVIStepAdditions Constants

Use these string constants to create lookupStrings to access the adapter-specific properties of a C/CVI step. The following example code sets the module pathname property of a C/CVI step:

TS_PropertySetValString(propObj, &errorInfo, Step_TSInfoProp "." TSInfo_StepAdditions "." CVIStep_ModulePathProp, 0, testFilePath);

- **CVIStep_FunctionNameProp** -- The name of the C function to call.
- **CVIStep_FunctionPrototypeProp** -- A number property that indicates the prototype to use when calling the test function. Use 0 for the Standard CVI TestExecutive prototype. Use 1 for the Extended CVI TestExecutive prototype. The CVI Test Executive used the Extended prototype for setup/cleanup and load/unload steps.
- **CVIStep_ModulePathProp** -- The pathname of the module that contains the function to call.
- **CVIStep_ModulePrjPathProp** -- The pathname of the LabWindows/CVI project file that creates the code module for the step. If the module is a dynamic link library or a static library, this property is required to enable the Edit Code command in the sequence editor. You can also specify this property for object files. The adapter ignores this property for C source files.
- **CVIStep_ModuleSrcPathProp** -- The pathname of the source file that contains the function to call. This property is required to enable the Edit Code command in the sequence editor.
- **CVIStep_ModuleTypeProp** -- The type of code module specified by the module path property. 0 = object file, 1 = C source file, 2 = DLL, 3 = static library or DLL import library
- **CVIStep_ParamsStringProp** -- The string to pass as the 'params' parameter for the Extended prototype. Steps that use the Standard prototype do not have a 'params' parameter.
- **CVIStep_SeqContextPassProp** -- A Boolean property that controls whether the adapter passes a sequence context to the code module.

DefaultModelCallbacks Constants

Use these string constants to specify the callbackName parameter of the SequenceFile.CreateCallbackOverrideSequence method.

- DefModCback_ModifyRptEntry
- DefModCback_PostUUT
- DefModCback_PostUUTLoop
- DefModCback_PreUUT
- DefModCback_PreUUTLoop
- DefModCback_ReportOptions
- DefModCback_TestReport

EditArgs

Use objects of the EditArgs class to pass information about the current state of the operator interface to methods such as Engine.NewExecution, Engine.ConstructToolMenus, and Engine.EvalToolMenuItemExprs. Objects of this class contain information about currently selected sequences, steps, sequence files, and executions. This selection information appears in the RunState.InitialSelection property of the sequence context.

Create objects of this class using Engine.NewEditArgs. Generally, you use this class only when writing an operator interface.

Methods

AddSelectedSequence	AddSelectedStep	AsPropertyObject
ClearSelectedSequences	ClearSelectedSteps	SetSelectedExecution
SetSelectedSequenceFile		

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

AddSelectedSequence Method

Syntax

EditArgs.**AddSelectedSequence**(sequence)

Purpose

Adds a sequence to the list of currently selected sequences.

Parameter	Type	Description
sequence	Sequence	

AddSelectedStep Method

Syntax

EditArgs.**AddSelectedStep**(step)

Purpose

Adds a step to the list of currently selected steps.

Parameter	Type	Description
step	Step	

AsPropertyObject Method

Syntax

EditArgs.**AsPropertyObject**()

Return Type

PropertyObject

Purpose

Returns the underlying PropertyObject that represents the EditArgs object. Use the PropertyObject to modify, add, or remove your own custom properties of the object.

Remarks

Do not use this function to remove or modify the dynamic properties that TestStand already provides for EditArgs objects. You can do this through the other methods of the EditArgs class.

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

ClearSelectedSequences Method

Syntax

EditArgs.**ClearSelectedSequences**()

Purpose

Clears the internal list of selected sequences.

ClearSelectedSteps Method

Syntax

EditArgs.**ClearSelectedSteps**()

Purpose

Clears the internal list of selected steps.

SetSelectedExecution Method

Syntax

EditArgs.**SetSelectedExecution**(execution)

Purpose

Sets an execution as the selected execution.

Parameter	Type	Description
execution	Execution	

SetSelectedSequenceFile Method

Syntax

EditArgs.**SetSelectedSequenceFile**(sequenceFile)

Purpose

Sets a sequence file as the selected sequence file.

Parameter	Type	Description
sequenceFile	SequenceFile	

Engine

Use the Engine class to create and access objects of other classes, to control executions, to display built-in dialog boxes, to implement a Tools menu, to find files and directories, and to invoke various utilities. Generally, you use this class only when writing an operator interface. When writing an operator interface, create the engine object directly using ActiveX. The engine class is registered as "TestStand.Engine" on the machine that has TestStand installed. To access the Engine object from a step, use the API to get the value of the Engine property from the sequence context or pass the sequence context property "RunState.Engine" as an Object parameter to your step.

Examples of creating the initial Engine object:

LabWindows/CVI:

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

```
CAObjHandle engineObj = 0;
TS_NewEngine(NULL, &engineObj);
```

Visual Basic:

```
'Just add TestStand Engine as a control on your main form.
```

Visual C/C++:

```
Engine engineObj;
engineObj.CreateDispatch("TestStand.Engine");
```

Properties

AlwaysGotoCleanupOnFailure	ApplicationIsEditor	AppMainHwnd
AutoLoginSystemUser	BinDirectory	BreakOnRTE
BreakpointsEnabled	BuiltinDataTypes	ConfigDirectory
CurrentUser	CustomDataTypes	DefaultAdapter
DisableResults	EnableUserPrivilegeChecking	ExecutionMask
Globals	IsUIMessageQueueEmpty	MajorVersion
MinorVersion	NumAdapters	RevisionVersion
StepTypes	TestStandDirectory	TracingEnabled
UIMessageDelay	UIMessageMinDelay	UIMessagePollingEnabled
VersionString		

Methods

AbortAll	BreakAll	CallFrontEndCallback
CheckExprSyntax	CommitGlobalsToDisk	ConstructToolMenus
CreateTempFile	CurrentUserHasPrivilege	DisplayAdapterConfigDialog
DisplayBrowseExprDialog	DisplayEditUserDialog	DisplayExternalViewerDialog
DisplayLoginDialog	DisplayLoopOnStepsDialog	DisplayNewUserDialog
DisplayOpenFileDialog	DisplayOptionsDialog	DisplayPreconditionDialog
DisplayRunTimeErrorDialog	DisplaySearchDirDialog	DisplaySeqFilePropDialog
DisplaySequencePropDialog	DisplayStepPropDialog	DisplayToolMenuDialog
EvalToolMenuItemExprs	FindFile	FindPath
GetAdapter	GetErrorString	GetNumToolMenuItems
GetNumToolMenus	GetResourceString	GetSequenceFile
GetStationModelSequenceFile	GetToolMenuItemInfo	GetToolMenuItemInfoWithID
GetUIMessage	GetUser	GetUserProfile
InvokeToolMenuItem	InvokeToolMenuItemWithID	LaunchExternalViewer
NewEditArgs	NewExecution	NewInteractiveArgs
NewPropertyObject	NewSequence	NewSequenceFile
NewStep	NewUser	NotifyEndOfModalDialog
NotifyStartOfModalDialog	RegisterUIMessageCallback	ReleaseSequenceFile
ReloadGlobals	ShutDown	TerminateAll
UIMessageEvent	UnloadAllModules	UserNameExists

See Also

SequenceContext.Engine

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

AlwaysGotoCleanupOnFailure Property

Syntax

Engine.**AlwaysGotoCleanupOnFailure**()

Data Type

Boolean

Purpose

When set to True, this property overrides the Sequence.GotoCleanupOnFailure setting.

Remarks

See the Sequence.GotoCleanupOnFailure topic for more information.

See Also

Sequence.GotoCleanupOnFailure

ApplicationIsEditor Property

Syntax

Engine.**ApplicationIsEditor**()

Data Type

Boolean

Purpose

Default setting is False. Set this property to True if you are using the TestStand API to write a sequence editor. If you are writing a run-time operator interface, keep the default setting, False.

AppMainHwnd Property

Syntax

Engine.**AppMainHwnd**()

Data Type

Long

Purpose

When you are developing a sequence editor or run-time operator interface, set this property to the value of the main Windows window handle if you want TestStand dialog boxes to be modal to your application. Depending on the development environment you use to create the user-interface, modal dialog boxes might not be possible. Failing to set this property ensures that the dialogs are non-modal and causes TestStand to create separate taskbar buttons for each dialog box.

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

AutoLoginSystemUser Property

Syntax

Engine.**AutoLoginSystemUser**()

Data Type

Boolean

Purpose

When this property is set to True, the loginlogout callback in TestStand uses the current user's login for the operating system as the login for TestStand when first launching an operator interface.

Remarks

If the user's login does not exist in TestStand, the login dialog box appears. If the user's login does exist, the user with that login is automatically logged in. TestStand performs no password authentication for an automatic login.

BinDirectory Property (Read Only)

Syntax

Engine.**BinDirectory**()

Data Type

String

Purpose

Returns the pathname of the TestStand Bin directory. The pathname does NOT contain a trailing backslash.

BreakOnRTE Property

Syntax

Engine.**BreakOnRTE**()

Data Type

Boolean

Purpose

Set to True to require the sequence editor or run-time operator interface to break on run-time errors. In most sequence editors and operator interfaces, breaking on a run-time error displays a dialog box that allows the user to choose how to proceed.

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

BreakpointsEnabled Property

Syntax
Engine.**BreakpointsEnabled**()

Data Type
Boolean

Purpose
Set this property to True to honor breakpoints. Set it to False to ignore them.

BuiltinDataTypes Property (Read Only)

Syntax
Engine.**BuiltinDataTypes**()

Data Type
PropertyObject

Purpose
Returns the PropertyObject that contains the built-in data types for the test station.

Remarks
Sequence editors typically use this property to display a list of available data types.

ConfigDirectory Property (Read Only)

Syntax
Engine.**ConfigDirectory**()

Data Type
String

Purpose
Returns the pathname of the TestStand configuration file directory. The pathname does NOT contain a trailing backslash.

CurrentUser Property

Syntax
Set *Engine*.**CurrentUser**()

Data Type
User

Purpose
Use this property to get or set the User object for the user who is currently logged in. It returns a NULL dispatch pointer or object reference when no user is currently logged in.

Remarks
After you obtain the User object, you can call the HasPrivilege method to determine whether the user has a specific privilege.

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

CustomDataTypes Property (Read Only)

Syntax
Engine.**CustomDataTypes**()

Data Type
PropertyObject

Purpose
Returns the PropertyObject that contains the custom data types for the test station.

Remarks
Sequence editors typically use this property to display a list of available data types.

DefaultAdapter Property

Syntax
Engine.**DefaultAdapter**()

Data Type
String

Purpose
Use this property to get or set the key name of the adapter to use by default when creating steps.

DisableResults Property

Syntax
Engine.**DisableResults**()

Data Type
Boolean

Purpose
When set to True, TestStand does not record results for any steps. When set to False, TestStand records results based on the setting of the RecordResult property of each individual step or based on the DisableResults property of the sequence.

EnableUserPrivilegeChecking Property

Syntax
Engine.**EnableUserPrivilegeChecking**()

Data Type
Boolean

Purpose
When set to False, the sequence editor or run-time operator interface does not check whether users have the privileges necessary for performing specific operations. Instead, all operations dependent on user privileges are always available.

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

ExecutionMask Property

Syntax
Engine.**ExecutionMask**()

Data Type
Long

Purpose
Use this property to set or get execution options. You specify the options using ExecutionMask constants. Specify multiple options by using the bitwise-OR operator.

Globals Property (Read Only)

Syntax
Engine.**Globals**()

Data Type
PropertyObject

Purpose
Returns the PropertyObject that contains the global variables for the test station.

Remarks
Sequence editors and run-time operator interfaces typically use this property to display globals or to allow users to edit them.

IsUIMessageQueueEmpty Property (Read Only)

Syntax
Engine.**IsUIMessageQueueEmpty**()

Data Type
Boolean

Purpose
If you are writing a sequence editor or run-time operator interface, use this property when polling for UIMessages to determine whether a message is pending in the queue. You must set the Engine.UIMessagePollingEnabled property to True to poll for UIMessages.

MajorVersion Property (Read Only)

Syntax
Engine.**MajorVersion**()

Data Type
Long

Purpose
The major version number for the current version of TestStand.

Remarks

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

The version number usually appears in the following format: MajorVersion.MinorVersion.RevisionVersion. For example, "1.0.0".

See Also
MinorVersion
RevisionVersion
VersionString

MinorVersion Property (Read Only)

Syntax
Engine.**MinorVersion**()

Data Type
Long

Purpose

The minor version number for the current version of TestStand.

Remarks

The version number usually appears in the following format: MajorVersion.MinorVersion.RevisionVersion. For example, "1.0.0".

See Also
MajorVersion
RevisionVersion
VersionString

NumAdapters Property (Read Only)

Syntax
Engine.**NumAdapters**()

Data Type
Long

Purpose

Returns the number of module adapters that are available for use with TestStand.

See Also
GetAdapter

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

RevisionVersion Property (Read Only)

Syntax

Engine.**RevisionVersion**()

Data Type

Long

Purpose

The revision version number for the current version of TestStand.

Remarks

The version number usually appears in the following format: MajorVersion.MinorVersion.RevisionVersion. For example, "1.0.0".

See Also

MajorVersion
MinorVersion
VersionString

StepTypes Property (Read Only)

Syntax

Engine.**StepTypes**()

Data Type

PropertyObject

Purpose

Returns the PropertyObject that contains the step types for the test station.

Remarks

Sequence editors typically use this property to display a list of step types from which users can select when they insert a new step in a sequence.

TestStandDirectory Property (Read Only)

Syntax

Engine.**TestStandDirectory**()

Data Type

String

Purpose

Returns the pathname of the directory in which you installed TestStand. The pathname does NOT contain a trailing backslash.

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

TracingEnabled Property

Syntax

Engine.**TracingEnabled**()

Data Type

Boolean

Purpose

When False, execution tracing is turned off for sequence editors and run-time operator interfaces.

UIMessageDelay Property

Syntax

Engine.**UIMessageDelay**()

Data Type

Long

Purpose

Set this property to specify how many milliseconds must pass before you receive the next UIMsg_Trace event. This property is useful for slowing down the posting of trace messages to the operator interface or sequence editor in order to allow time for processing mouse events and/or to display the trace of an execution more slowly. The value of this property can never be less than the value of the UIMessageMinDelay property. If set to a lesser value, UIMessageDelay is automatically reset to the value of UIMessageMinDelay.

See Also

UIMessageMinDelay
UIMessagePollingEnabled
GetUIMessage
RegisterUIMessageCallback
UIMessageEvent

UIMessageMinDelay Property

Syntax

Engine.**UIMessageMinDelay**()

Data Type

Long

Purpose

Default value is zero. Use this property to specify the minimum value allowed for the UIMessageDelay property for your sequence editor or run-time operator interface.

See Also

UIMessageDelay
UIMessagePollingEnabled
GetUIMessage
RegisterUIMessageCallback
UIMessageEvent

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

UIMessagePollingEnabled Property

Syntax

Engine.**UIMessagePollingEnabled**()

Data Type

Boolean

Purpose

Default value is False. Set this property to True when writing an operator interface which uses polling rather than events to get UIMessages from the TestStand.

Remarks

To poll, call IsUIMessageQueueEmpty. If IsUIMessageQueueEmpty returns False, call GetUIMessage to retrieve the next UIMessage. Release your reference to the previous UIMessage before getting the next one.

See Also

UIMessageDelay
UIMessageMinDelay
GetUIMessage
RegisterUIMessageCallback
UIMessageEvent

VersionString Property (Read Only)

Syntax

Engine.**VersionString**()

Data Type

String

Purpose

Returns the full version of the TestStand engine as a string.

Remarks

Use this string for display purposes. Use the other version properties for conditional code.

See Also

MajorVersion
MinorVersion
RevisionVersion

AbortAll Method

Syntax

Engine.**AbortAll**()

Purpose

Aborts all existing executions.

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

BreakAll Method

Syntax

Engine.**BreakAll**()

Purpose

Suspends all existing executions.

CallFrontEndCallback Method

Syntax

Engine.**CallFrontEndCallback**(sequenceName, argumentList)

Return Type

Execution

Purpose

Calls the Front End callback sequence that sequenceName specifies.

Remarks

Front End callbacks are callbacks that sequence editor and run-time operator interface programs call. Front End callbacks allow you to customize the sequence editors and operator interfaces on your system. Each sequence editor or operator interface chooses which Front End callbacks it calls. The Front End callbacks that National Instruments defines are located in the FrontEndCallbacks.seq file in the TestStand\Components\NI\Callbacks\FrontEnd directory. Currently, National Instruments defines only one Front End callback: LoginLogout. You can override the National Instruments version of LoginLogout or add your own Front End callbacks in the FrontEndCallbacks.seq file that is located in the TestStand\Components\User\Callbacks\FrontEnd directory.

Return Value

The Execution object of the newly created execution.

Parameter	Type	Description
sequenceName	String	The name of the front end sequence to call.
argumentList	PropertyObject	A propertyObject that contains the parameters to pass to the sequence. Each subproperty of the propertyObject represents a parameter to the sequence. The subproperties must appear in the same order as the sequence parameters.

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

CheckExprSyntax Method

Syntax

Engine.**CheckExprSyntax**(expression, errorDescription, startErrPos, endErrPos)

Return Type

Boolean

Purpose

Checks the syntax of the expression parameter and returns error information.

Return Value

Returns True if the syntax is correct. Returns False if the expression contains a syntax error. If it returns False, check the errorDescription, startErrPos, and endErrPos for more information.

Parameter	Type	Description
expression	String	The expression for which to check the syntax.
errorDescription	String	When a syntax error exists, this parameter contains an error message describing the type of error.
startErrPos	Long	When a syntax error exists, this parameter returns the index of the location in the string where the error begins.
endErrPos	Long	When a syntax error exists, this parameter returns the index of the location in the string of the first character beyond the syntax error.

CommitGlobalsToDisk Method

Syntax

Engine.**CommitGlobalsToDisk**(promptOnSaveConflicts = True)

Purpose

Saves the current state of the Globals to disk.

Parameter	Type	Description
promptOnSaveConflicts	Boolean	TestStand writes the globals to disk only if they change after TestStand has loaded or saved them. If another application changes the globals on disk after TestStand has loaded or saved them, the action that TestStand takes depends on the value you pass in this parameter. If you pass True, TestStand gives the user the option to overwrite the changes that the other application made. If you pass False, TestStand saves the globals without prompting the user to resolve any conflicts between the changes in memory and the changes on disk. This parameter has a default value of True.

See Also

Globals

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

ConstructToolMenus Method

Syntax

Engine.**ConstructToolMenus**([editArgsParam])

Return Type

Long

Purpose

Creates the internal data for the tool menu that appears in operator interfaces and sequence editors.

Remarks

Call this function to initialize the tool menu and its submenus for the first time and whenever DisplayToolMenuDialog modifies the tool menu or its submenus.

Return Value

The number of tool menus that exist. This number includes the tool menu and its submenus.

Parameter	Type	Description
editArgsParam	Variant	An optional EditArgs parameter in which you specify which items in the operator interface are currently selected. The method uses this information to calculate the name and enabled state of each menu item by evaluating its "name" and "enabled" expressions.

See Also

GetNumToolMenus
GetNumToolMenuItems
GetToolMenuItemInfo
GetToolMenuItemInfoWithID
InvokeToolMenuItem
InvokeToolMenuItemWithID
EvalToolMenuItemExprs
DisplayToolMenuDialog
EditArgs

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

CreateTempFile Method

Syntax

Engine.**CreateTempFile**(baseName, extension, directory)

Return Type

String

Purpose

Creates a unique temporary file with a specific basename and extension. When TestStand shuts down, it deletes the file automatically.

Return Value

The pathname of the temporary file.

Parameter	Type	Description
baseName	String	The base name of the temporary file. Other characters are added to the name to make it unique if necessary.
extension	String	The file extension for the temporary file.
directory	String	The directory in which to store the temporary file. You can include a trailing backslash, but you do not have to. Pass an empty string to use the <TestStand>\Bin\Temp directory.

CurrentUserHasPrivilege Method

Syntax

Engine.**CurrentUserHasPrivilege**(privilegeName)

Return Type

Boolean

Purpose

Confirms whether the current user has a specific privilege.

Remarks

Returns True when the privilege property is True, when the privilege property of any group that contains the privilege is True, or when privilege checking is disabled.

Return Value

True if the current user has the privilege.

Parameter	Type	Description
privilegeName	String	The name of the privilege to check. You can specify the name of any privilege property in the user privileges property tree, including a group privilege. You do not have to include the names of all the containing groups. If you specify only the base name of a privilege, and more than one group contains a privilege with that name, the method returns the value of the first privilege it finds with that name.

See Also

UserPrivileges

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

DisplayAdapterConfigDialog Method

Syntax

Engine.**DisplayAdapterConfigDialog**(dlgTitle, adapterSelectorReadOnly, adapterCfgReadOnly, hideAdapterSelector, modalToAppMainWind)

Return Type

Boolean

Purpose

Displays a dialog box in which you can edit adapter specific settings and select the default adapter.

Return Value

True if the dialog box modifies adapter settings.

Parameter	Type	Description
dlgTitle	String	Specifies the title of the dialog box. Pass an empty string to use the default title for the dialog box.
adapterSelectorReadOnly	Boolean	Pass True if you want a read-only version of the adapter selector in the dialog box. The adapter selector is used to select the default adapter for the Engine.
adapterCfgReadOnly	Boolean	Pass True if you want read-only versions of the adapter configuration controls in the dialog box.
hideAdapterSelector	Boolean	Pass True to hide the adapter selector control of the dialog box. This is useful for operator interfaces where the ability to select a default adapter is not needed.
modalToAppMainWind	Boolean	If you want the dialog to be modal with respect to the window handle that the Engine.AppMainHwnd property stores, pass True for this parameter. Pass False to create a non-modal version of the dialog box.

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

DisplayBrowseExprDialog Method

Syntax

Engine.**DisplayBrowseExprDialog**(dlgTitle, sequenceContextParam, expressionIn, selectionStartIn, selectionEndIn, initialVariableName, usesOnlyLF, modalToAppMainWind, expressionOut, selectionStartOut, selectionEndOut)

Return Type

Boolean

Purpose

Displays a dialog box that lets you construct an expression string using variables, properties, constants, operators, and functions.

Return Value

True if you click the OK button in the dialog. False if you cancel.

Parameter	Type	Description
dlgTitle	String	Specifies the title of the dialog box. Pass an empty string to use the default title for the dialog box.
sequenceContextParam	SequenceContext	Pass the sequence context into which to browse. You can use the SequenceFile.NewEditContext method to get an edit time sequence context from a sequence file.
expressionIn	String	Pass the initial expression string for the dialog box.
selectionStartIn	Long	Pass the zero-based index of the location in the initial expression where the selected text begins. If you want a cursor instead of selected text, pass the same index for selectionEndIn.
selectionEndIn	Long	Pass the index of the location in the initial expression of the first character beyond the selected text. If you want a cursor instead of selected text, pass the same index for selectionStartIn.
initialVariableName	String	Pass the name of the initial variable or property to show as selected in the treeview portion of the dialog box.
usesOnlyLF	Boolean	If the initial expression string uses only a linefeed ("\n") to indicate the end of a line of text, pass True for this parameter. If the string uses a carriage return/linefeed combination ("\r\n"), pass False.
modalToAppMainWind	Boolean	If you want the dialog to be modal with respect to the window handle that the Engine.AppMainHwnd property stores, pass True for this parameter. Pass False to create a non-modal version of the dialog box.
expressionOut	String	Returns the resulting expression from the dialog box when the user clicks on OK.
selectionStartOut	Long	Returns the index of the start of the selected text in the resulting expression when the user clicks on OK.
selectionEndOut	Long	Returns the index of the first character beyond the selected text in the resulting expression when the user clicks on OK.

See Also

AppMainHwnd
SequenceFile.NewEditContext

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

DisplayEditUserDialog Method

Syntax

Engine.**DisplayEditUserDialog**(dlgTitle, userObject, modalToAppMainWind)

Return Type

Boolean

Purpose

Displays a dialog box in which you can edit the user information for a specific user.

Remarks

You can edit the name, comment, and password.

Return Value

True if you click on the OK button in the dialog box. False if you cancel.

Parameter	Type	Description
dlgTitle	String	Specifies the title of the dialog box. Pass an empty string to use the default title for the dialog box.
userObject	User	Pass the User object that you want to edit.
modalToAppMainWind	Boolean	If you want the dialog box to be modal with respect to the window handle that the Engine.AppMainHwnd property stores, pass True for this parameter. Pass False to create a non-modal version of the dialog box.

DisplayExternalViewerDialog Method

Syntax

Engine.**DisplayExternalViewerDialog**(dlgTitle, readOnly, modalToAppMainWind)

Return Type

Boolean

Purpose

Displays a dialog box in which you can edit the external file viewer settings.

Remarks

The external viewer settings specify the external viewer applications you use to view report files of various formats. Formats include text (.txt) and HTML (.html) files.

Return Value

True if you click on the OK button in the dialog box. False if you cancel.

Parameter	Type	Description
dlgTitle	String	Specifies the title of the dialog box. Pass an empty string to use the default title for the dialog box.
readOnly	Boolean	Pass True if you want a read-only version of the dialog box.
modalToAppMainWind	Boolean	If you want the dialog box to be modal with respect to the window handle that the Engine.AppMainHwnd property stores, pass True for this parameter. Pass False to create a non-modal version of the dialog box.

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

See Also

LaunchExternalViewer

DisplayLoginDialog Method

Syntax

Engine.**DisplayLoginDialog**(dlgTitle, initialLoginName, initialPassword, modalToAppMainWind, userObject)

Return Type

Boolean

Purpose

Displays a login dialog box. A dropdown list on the dialog box contains the login names of all the current TestStand users.

Remarks

The LoginLogout Front End callback calls this method.

Return Value

True if you click on the OK button in the dialog box. False if you cancel.

Parameter	Type	Description
dlgTitle	String	Specifies the title of the dialog box. Pass an empty string to use the default title for the dialog box.
initialLoginName	String	The initial login name that you want to appear in the dialog box.
initialPassword	String	The initial password you want to appear in the dialog box. It appears on screen as all asterisks (*).
modalToAppMainWind	Boolean	If you want the dialog box to be modal with respect to the window handle that the Engine.AppMainHwnd property stores, pass True for this parameter. Pass False to create a non-modal version of the dialog box.
userObject	User	Returns the User object that represents the user who logged in.

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

DisplayLoopOnStepsDialog Method

Syntax

Engine.**DisplayLoopOnStepsDialog**(dlgTitle, selectedStep, modalToAppMainWnd, loopCountValue, stopExpressionValue)

Return Type

Boolean

Purpose

Displays a dialog box that prompts the user to provide interactive execution information required for the Loop On Steps command.

Return Value

True if you click on the OK button in the dialog box. False if you cancel.

Parameter	Type	Description
dlgTitle	String	Specifies the title of the dialog box. Pass an empty string to use the default title for the dialog box.
selectedStep	Step	Pass a reference to the first step that is currently selected. The dialog box uses this step to display property information if the user browses to create the 'stop on' expression.
modalToAppMainWnd	Boolean	If you want the dialog box to be modal with respect to the window handle that the Engine.AppMainHwnd property stores, pass True for this parameter. Pass False to create a non-modal version of the dialog box.
loopCountValue	Long	The value the of loop count that the user specifies in the dialog box. A value of -1 specifies an infinite loop.
stopExpressionValue	String	The value of the stop expression that the user specifies in the dialog box. An empty string specifies that the stop expression is not used.

DisplayNewUserDialog Method

Syntax

Engine.**DisplayNewUserDialog**(dlgTitle, modalToAppMainWind, userObject)

Return Type

Boolean

Purpose

Displays a dialog box for creating a new user.

Return Value

True if you click on the OK button in the dialog box. False if you cancel.

Parameter	Type	Description
dlgTitle	String	Specifies the title of the dialog box. Pass an empty string to use the default title for the dialog box.
modalToAppMainWind	Boolean	If you want the dialog box to be modal with respect to the window handle that the Engine.AppMainHwnd property stores, pass True for this parameter. Pass False to create a non-modal version of the dialog box.
userObject	User	Returns the User object that represents the newly created user.

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

DisplayOpenFileDialog Method

Syntax

Engine.**DisplayOpenFileDialog**(dlgTitle, okButtonText, initialPath, modalToAppMainWind, selectedPath, absolutePath, openFileDialogFlags = 0, defaultExtension = "", win32Flags = OFN_HIDEREADONLY | OFN_FILEMUSTEXIST, fileFilter = "" [, currentSequenceFile])

Return Type

Boolean

Purpose

Displays a dialog box in which the user can select a file.

Remarks

The user can choose whether the dialog box returns an absolute or relative pathname for the file. The dialog box can return a relative pathname only when the file is located under one of the TestStand search directories. If the file is not located under any of the TestStand search directories and the user requests that the dialog box return a relative pathname, the dialog box prompts the user to add the name of the directory that contains the file to the list of TestStand search directories. You can pass a sequence file object to include its directory among the list of search directories that are valid for a relative pathname.

Return Value

True if you click on the OK button in the dialog box. False if you cancel.

Parameter	Type	Description
dlgTitle	String	Specifies the title of the dialog box. Pass an empty string to use the default title for the dialog box.
okButtonText	String	Pass a string to display on the OK button, or pass an empty string to use the default.
initialPath	String	Pass the path of the directory that you want the user to browse initially. Pass an empty string to specify the current directory.
modalToAppMainWind	Boolean	If you want the dialog box to be modal with respect to the window handle that the Engine.AppMainHwnd property stores, pass True for this parameter. Pass False to create a non-modal version of the dialog box.
selectedPath	String	Returns the pathname that the user specifies in the dialog box. The path can be relative.
absolutePath	String	Returns the absolute pathname of the file the user selects.
openFileDialogFlags	Long	Pass 0 for the default behavior, or pass one or more OpenFileDialogFlags constants. Use the bitwise-OR operator to specify multiple flags. This parameter has a default value of 0.
defaultExtension	String	The extension to append to the pathname if the user specifies no extension. Do not include the period in the extension. This parameter has a default value of "".
win32Flags	Long	Use this parameter to specify Windows file dialog flags. See OPENFILENAMEFlags for more information. This parameter has a default value of OFN_HIDEREADONLY | OFN_FILEMUSTEXIST.
fileFilter	String	The fileFilter string specifies the extensions that files must have in order to appear in the dialog box. Each filter that the string contains has two parts. The first part is a descriptive name for the filter and the second part lists the extensions that the file can have. Specify multiple extensions using semi-colon (;) as a delimiter, and end each section with a vertical bar (|). End the string with a final vertical bar (|). The following example illustrates this format:

"Sequence Files (*.seq)|*.seq|Report Files (*.txt;*.htm;*.html)|*.txt;*.htm;*.html||" This parameter has a default value of "".

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

currentSequenceFile	Variant	Pass a SequenceFile object if you want to include the directory of the sequence file in the list of search directories that are valid for a relative pathname.

See Also

OpenFileDialogFlags

DisplayOptionsDialog Method

Syntax

Engine.**DisplayOptionsDialog**(dlgTitle, readOnly, modalToAppMainWind)

Return Type

Boolean

Purpose

Displays the Station Options dialog box.

Remarks

The Station Options dialog box contains numerous settings that affect execution, user privileges, and process models.

Return Value

True if you click on the OK button in the dialog box. False if you cancel.

Parameter	Type	Description
dlgTitle	String	Specifies the title of the dialog box. Pass an empty string to use the default title for the dialog box.
readOnly	Boolean	Pass True if you want a read-only version of the dialog box.
modalToAppMainWind	Boolean	If you want the dialog box to be modal with respect to the window handle that the Engine.AppMainHwnd property stores, pass True for this parameter. Pass False to create a non-modal version of the dialog box.

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

DisplayPreconditionDialog Method

Syntax

Engine.**DisplayPreconditionDialog**(dlgTitle, sequence, readOnly, modalToAppMainWind [, initialStep])

Return Type

Boolean

Purpose

Displays a dialog box in which you can edit the preconditions for a step or for all the steps in a sequence..

Return Value

True if you click on the OK button in the dialog box. False if you cancel.

Parameter	Type	Description
dlgTitle	String	Specifies the title of the dialog box. Pass an empty string to use the default title for the dialog box.
sequence	Sequence	Pass the sequence in which to edit preconditions.
readOnly	Boolean	Pass True if you want a read-only version of the dialog.
modalToAppMainWind	Boolean	If you want the dialog to be modal with respect to the window handle that the Engine.AppMainHwnd property stores, pass True for this parameter. Pass False to create a non-modal version of the dialog.
initialStep	Variant	Pass a Step object if you want the dialog box to display only the preconditions for a particular step. This parameter is optional.

DisplayRunTimeErrorDialog Method

Syntax

Engine.**DisplayRunTimeErrorDialog**(dlgTitle, errorMessage, inCleanupStepGroup, modalToAppMainWind, displayOnNextError, suspendExecution, runTimeErrorAction)

Purpose

Displays a run-time error dialog box.

Remarks

Sequence editor and run-time operator interface programs typically call this function in response to a UIMsg_BreakOnRunTimeError event. The dialog box allows the user to specify how the execution is to proceed.

Parameter	Type	Description
dlgTitle	String	Specifies the title of the dialog box. Pass an empty string to use the default title for the dialog box.
errorMessage	String	Pass the error message to display in the dialog.
inCleanupStepGroup	Boolean	Pass True if the run-time error occurred in the Cleanup step group of the sequence. This information affects the list of options that the dialog displays to the user.
modalToAppMainWind	Boolean	If you want the dialog to be modal with respect to the window handle that the Engine.AppMainHwnd property stores, pass True for this parameter. Pass False to create a non-modal version of the dialog.
displayOnNextError	Boolean	Returns whether the user wants this dialog to appear again if another run-time error occurs in the current execution.

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

suspendExecution	Boolean	Returns whether the user wants the current execution to suspend at the location of the run-time error.
runTimeErrorAction	RTEOptions	Returns the option the user selects in the dialog to specify how execution is to proceed.

DisplaySearchDirDialog Method

Syntax

Engine.**DisplaySearchDirDialog**(dlgTitle, readOnly, modalToAppMainWind)

Return Type

Boolean

Purpose

Displays a dialog box in which you can edit the list of TestStand search directories.

Return Value

True if you click on the OK button in the dialog box. False if you cancel.

Parameter	Type	Description
dlgTitle	String	Specifies the title of the dialog box. Pass an empty string to use the default title for the dialog box.
readOnly	Boolean	Pass True if you want a read-only version of the dialog box.
modalToAppMainWind	Boolean	If you want the dialog box to be modal with respect to the window handle that the Engine.AppMainHwnd property stores, pass True for this parameter. Pass False to create a non-modal version of the dialog box.

DisplaySeqFilePropDialog Method

Syntax

Engine.**DisplaySeqFilePropDialog**(dlgTitle, sequenceFileParam, readOnly, modalToAppMainWind, showViewContentsBtn, viewContents)

Return Type

Boolean

Purpose

Displays a dialog box in which you can edit the properties of a sequence file.

Remarks

You can edit the load/unload options, model options, and other options.

Return Value

True if you click on the OK button in the dialog box. False if you cancel.

Parameter	Type	Description
dlgTitle	String	Specifies the title of the dialog box. Pass an empty string to use the default title for the dialog box.
sequenceFileParam	SequenceFile	Pass a SequenceFile object to specify which sequence file you want to edit.
readOnly	Boolean	Pass True If you want a read-only version of the dialog box.

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

modalToAppMainWind	Boolean	If you want the dialog box to be modal with respect to the window handle that the Engine.AppMainHwnd property stores, pass True for this parameter. Pass False to create a non-modal version of the dialog box.
showViewContentsBtn	Boolean	Pass True to show a View Contents button in the dialog box. If you pass True, the viewContents parameter of this method returns True when a user clicks on the View Contents button.
viewContents	Boolean	Returns True if a user clicks on the View Contents button of the dialog box.

DisplaySequencePropDialog Method

Syntax

Engine.**DisplaySequencePropDialog**(dlgTitle, sequence, readOnly, modalToAppMainWind, showViewContentsBtn, viewContents)

Return Type

Boolean

Purpose

Displays a dialog box in which you can edit the properties of a sequence.

Return Value

True if you click the OK button in the dialog. False if you cancel.

Parameter	Type	Description
dlgTitle	String	Specifies the title of the dialog. Pass an empty string to use the default title for the dialog.
sequence	Sequence	Pass the Sequence object that contains the properties you want to edit.
readOnly	Boolean	Pass True if you want a read-only version of the dialog.
modalToAppMainWind	Boolean	If you want the dialog to be modal with respect to the window handle that the Engine.AppMainHwnd property stores, pass True for this parameter. Pass False to create a non-modal version of the dialog.
showViewContentsBtn	Boolean	Pass True to show a View Contents button in the dialog. If you pass True, the viewContents parameter of this method will return True if the dialog's user presses the View Contents button.
viewContents	Boolean	Returns True if the user presses the View Contents button of the dialog.

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

DisplayStepPropDialog Method

Syntax

Engine.**DisplayStepPropDialog**(dlgTitle, step, readOnly, modalToAppMainWind, showViewContentsBtn, viewContents, modifiedStep)

Return Type

Boolean

Purpose

Displays a dialog box in which you can edit the properties of a step.

Return Value

True if you click on the OK button in the dialog box. False if you cancel.

Parameter	Type	Description
dlgTitle	String	Specifies the title of the dialog box. Pass an empty string to use the default title for the dialog box.
step	Step	Pass the Step object that contains the properties you want to edit.
readOnly	Boolean	Pass True if you want a read-only version of the dialog box.
modalToAppMainWind	Boolean	If you want the dialog box to be modal with respect to the window handle that the Engine.AppMainHwnd property stores, pass True for this parameter. Pass False to create a non-modal version of the dialog box.
showViewContentsBtn	Boolean	Pass True to show a View Contents button in the dialog box. If you pass True, the viewContents parameter of this method returns True when a user clicks on the View Contents button.
viewContents	Boolean	Returns True if the user presses the View Contents button of the dialog box.
modifiedStep	Boolean	Returns True if the step was modified.

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

DisplayToolMenuDialog Method

Syntax

Engine.**DisplayToolMenuDialog**(dlgTitle, readOnly, modalToAppMainWind)

Return Type

Boolean

Purpose

Displays a dialog box in which you can edit the tool menu items.

Return Value

True if you click on the OK button in the dialog box. False if you cancel.

Parameter	Type	Description
dlgTitle	String	Specifies the title of the dialog box. Pass an empty string to use the default title for the dialog box.
readOnly	Boolean	Pass True if you want a read-only version of the dialog box.
modalToAppMainWind	Boolean	If you want the dialog box to be modal with respect to the window handle that the Engine.AppMainHwnd property stores, pass True for this parameter. Pass False to create a non-modal version of the dialog box.

EvalToolMenuItemExprs Method

Syntax

Engine.**EvalToolMenuItemExprs**([editArgsParam])

Purpose

Reevalutes the "enabled" expressions and "name" expressions of all the items in the Tools menu.

Remarks

You can extract the new "enabled" and "name" values for a particular Tools menu item by calling the GetToolMenuItemInfo or GetToolMenuItemInfoWithID method.

Parameter	Type	Description
editArgsParam	Variant	An optional EditArgs parameter in which you indicate which items in the operator interface are currently selected.

See Also

ConstructToolMenus
GetNumToolMenus
GetNumToolMenuItems
GetToolMenuItemInfo
GetToolMenuItemInfoWithID
InvokeToolMenuItem
InvokeToolMenuItemWithID
DisplayToolMenuDialog
EditArgs

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

FindFile Method

Syntax

Engine.**FindFile**(fileToFind, absolutePath, userCancelled, promptOption = FindFile_PromptHonorUserPreference, srchListOption = FindFile_AddDirToSrchList_Ask, isCommand = False [, currentSequenceFile])

Return Type

Boolean

Purpose

Searches for a file in the TestStand search directories using a simple filename or relative pathname.

Return Value

True if the file is found.

Parameter	Type	Description
fileToFind	String	A string containing the simple filename or relative pathname of the file to search for.
absolutePath	String	Returns the absolute pathname of the file if the file is found.
userCancelled	Boolean	Returns True if the user cancels the find operation.
promptOption	FindFilePromptOptions	Specifies whether to prompt the user if the file is not found initially. This parameter has a default value of FindFile_PromptHonorUserPreference.
srchListOption	FindFileSearchListOptions	Specifies file search options. This parameter has a default value of FindFile_AddDirToSrchList_Ask.
isCommand	Boolean	If this flag is True and fileToFind has NO file extension then FindFile searches for files with the same basename that end in the common command extensions: .exe, .com, and .bat. This parameter has a default value of False.
currentSequenceFile	Variant	If you pass a SequenceFile object for this parameter, FindFile initially searches the directory where the sequence file is stored.

See Also

FindPath

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

FindPath Method

Syntax

Engine.**FindPath**(pathToFind, absolutePath, statusFlag [, currentSequenceFile])

Return Type

Boolean

Purpose

Searches for a directory under the TestStand search directories using a simple directory name or relative pathname.

Return Value

True if the directory is found.

Parameter	Type	Description
pathToFind	String	A string containing the simple directory name or relative pathname of the directory to search for.
absolutePath	String	Returns the absolute pathname of the directory if it is found.
statusFlag	FindPathStatusValues	Returns more information about the directory.
currentSequenceFile	Variant	If you pass a SequenceFile object for this parameter, FindPath initially searches the directory where the sequence file is stored.

See Also

FindFile

GetAdapter Method

Syntax

Engine.**GetAdapter**(adapterIndex)

Return Type

Adapter

Purpose

Returns a module Adapter object from the list of module adapters.

Parameter	Type	Description
adapterIndex	Long	A zero-based index specifying which Adapter object to return. You can obtain the number of Adapter objects from the NumAdapters property.

See Also

NumAdapters

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

GetErrorString Method

Syntax

Engine.**GetErrorString**(errorCode, errorString)

Return Type

Boolean

Purpose

Returns an error description string that corresponds to a specific TSError code.

Return Value

Returns False when the errorCode is not a TSError.

Parameter	Type	Description
errorCode	TSError	The error code to describe.
errorString	String	Returns a description of the errorCode. If the error code is not a TSError, returns the string "User defined error code."

GetNumToolMenuItems Method

Syntax

Engine.**GetNumToolMenuItems**(menuIndex)

Return Type

Long

Purpose

Returns the number of menu items that a specific tool menu contains.

Return Value

The number of tool menu items.

Parameter	Type	Description
menuIndex	Long	The zero-based index of the tool menu from which to get the number of tool menu items.

See Also

ConstructToolMenus
EvalToolMenuItemExprs
GetNumToolMenus
GetToolMenuItemInfo
GetToolMenuItemInfoWithID
InvokeToolMenuItem
InvokeToolMenuItemWithID
DisplayToolMenuDialog

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

GetNumToolMenus Method

Syntax

Engine.**GetNumToolMenus**()

Return Type

Long

Purpose

Returns the number of tool menus.

Remarks

The first tool menu is the top-level menu. The rest of the menus are submenus.

Return Value

The number of tool menus.

See Also

ConstructToolMenus
EvalToolMenuItemExprs
GetNumToolMenuItems
GetToolMenuItemInfo
GetToolMenuItemInfoWithID
InvokeToolMenuItem
InvokeToolMenuItemWithID
DisplayToolMenuDialog

GetResourceString Method

Syntax

Engine.**GetResourceString**(category, symbol [, defaultString] [, found])

Return Type

String

Purpose

Returns a string from the TestStand string resource file for the current language. You specify the string by category and symbol, which are the section and item labels for the string in the string resource file.

Remarks

The string resource files are located in the TestStand\Components\NI\Language and TestStand\Components\User\Language directories. You can customize the strings in the resource files in the TestStand\Components\NI\Language directory as follows: add a section and item with the same name to the corresponding file in the TestStand\Components\User\Language directory.

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

Return Value

The resource string.

Parameter	Type	Description
category	String	The category or section label.
symbol	String	The symbol or item label.
defaultString	Variant	This parameter is optional. Pass a default string to return when the string resource file does not contain an entry with the category and symbol that you specify. If you do not specify this parameter and the string resource file does not contain the entry, this function returns the string "String not found in table."
found	Variant	This parameter is optional. Returns the Boolean value True if the string is found in the resource file, and False if the string is not found in the resource file.

GetSequenceFile Method

Syntax

Engine.**GetSequenceFile**(sequenceFilePath, getSeqFileFlags = GetSeqFile_OperatorInterfaceFlags)

Return Type

SequenceFile

Purpose

Returns the SequenceFile object for the sequence file that the sequenceFilePath parameter specifies.

Remarks

Loads the sequence file from disk if necessary. Call ReleaseSequenceFile on this object before releasing it.

Return Value

A sequence file object.

Parameter	Type	Description
sequenceFilePath	String	The pathname of the sequence file.
getSeqFileFlags	Long	Pass one or more GetSeqFileFlags constants. Use the bitwise-OR operator to specify multiple get sequence file flags. This parameter has a default value of GetSeqFile_OperatorInterfaceFlags.

See Also

GetSeqFileFlags
ReleaseSequenceFile

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

GetStationModelSequenceFile Method

Syntax

Engine.**GetStationModelSequenceFile**(modelDescriptionString)

Return Type

SequenceFile

Purpose

Returns a reference to the process model sequence file that TestStand associates with the test station. Release this reference when you are done using it.

Parameter	Type	Description
modelDescriptionString	String	Returns a string that describes the process model file.

See Also

SequenceFile.GetModelSequenceFile

GetToolMenuItemInfo Method

Syntax

Engine.**GetToolMenuItemInfo**(menuIndex, itemIndex, itemText, subMenuIndex, enabled, uniqueItemID)

Purpose

Returns information about a tool menu item that you specify with a menu index and item index.

Remarks

The information returned is useful for constructing a tool menu in an operator interface.

Parameter	Type	Description
menuIndex	Long	A zero-based index that specifies a tool menu or submenu.
itemIndex	Long	A zero-based index that specifies an item within the menu or submenu.
itemText	String	Returns the text to display in the menu item.
subMenuIndex	Long	If the item is a submenu, this parameter returns the zero-based menu index of the tool menu for the item. Otherwise, it returns -1.
enabled	Boolean	If True, you should enable the menu item. If False, disable it.
uniqueItemID	Long	Returns a unique id to identify the menu item. Use this id to invoke the tool menu item or to get the menu item information again at a later time.

See Also

ConstructToolMenus
EvalToolMenuItemExprs
GetNumToolMenuItems
GetNumToolMenus
GetToolMenuItemInfoWithID
InvokeToolMenuItem
InvokeToolMenuItemWithID
DisplayToolMenuDialog

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

GetToolMenuItemInfoWithID Method

Syntax

Engine.**GetToolMenuItemInfoWithID**(uniqueID, itemText, subMenuIndex, enabled)

Purpose

Returns information about a tool menu item that you specify with a unique menu ID that you obtain from GetToolMenuItemInfo.

Remarks

The information returned is useful for constructing a tool menu in an operator interface.

Parameter	Type	Description
uniqueID	Long	A tool menu item ID that you obtain from GetToolMenuItemInfo.
itemText	String	Returns the text to display in the menu item.
subMenuIndex	Long	If the item is a submenu, this parameter returns the zero-based menu index of the tool menu for the item. Otherwise, it returns -1.
enabled	Boolean	If True, you should enable the menu item. If False, disable it.

See Also

ConstructToolMenus
EvalToolMenuItemExprs
GetNumToolMenuItems
GetNumToolMenus
GetToolMenuItemInfo
InvokeToolMenuItem
InvokeToolMenuItemWithID
DisplayToolMenuDialog

GetUIMessage Method

Syntax

Engine.**GetUIMessage**()

Return Type

UIMessage

Purpose

When polling for UIMessages, use this method to get the next available message from the queue.

See Also

UIMessageDelay
UIMessageMinDelay
UIMessagePollingEnabled
RegisterUIMessageCallback
UIMessageEvent

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

GetUser Method

Syntax

Engine.**GetUser**(loginName)

Return Type

User

Purpose

Obtains the User object that contains a specific login name. Returns NULL if no User object has the specific login name.

Parameter	Type	Description
loginName	String	The login name to search for.

See Also

User

GetUserProfile Method

Syntax

Engine.**GetUserProfile**(userProfileName)

Return Type

User

Purpose

Returns a reference to a User object that represents a specific user profile. Returns NULL if the user profile you specify does not exist.

Remarks

The user profile is a template for an actual user. Pass the user profile object to the Engine.NewUser method to create a new user. Release your reference to the user profile object when you finish using it.

Parameter	Type	Description
userProfileName	String	Specifies the name of the user profile to return.

See Also

NewUser

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

InvokeToolMenuItem Method

Syntax

Engine.**InvokeToolMenuItem**(menuIndex, itemIndex)

Purpose

Executes the action associated with a tool menu item. You specify the menu item with a menu index and an item index.

Parameter	Type	Description
menuIndex	Long	A zero-based index that specifies a tool menu or submenu.
itemIndex	Long	A zero-based index that specifies an item within the menu or submenu.

See Also

ConstructToolMenus
EvalToolMenuItemExprs
GetNumToolMenuItems
GetNumToolMenus
GetToolMenuItemInfo
GetToolMenuItemInfoWithID
InvokeToolMenuItemWithID
DisplayToolMenuDialog

InvokeToolMenuItemWithID Method

Syntax

Engine.**InvokeToolMenuItemWithID**(uniqueItemID)

Purpose

Executes the action associated with a tool menu item. You specify the item with a unique item ID that you obtain from GetToolMenuItemInfo.

Parameter	Type	Description
uniqueItemID	Long	A tool menu item ID that you obtain from GetToolMenuItemInfo.

See Also

ConstructToolMenus
EvalToolMenuItemExprs
GetNumToolMenuItems
GetNumToolMenus
GetToolMenuItemInfo
GetToolMenuItemInfoWithID
InvokeToolMenuItem
DisplayToolMenuDialog

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

LaunchExternalViewer Method

Syntax

Engine.**LaunchExternalViewer**(filePath)

Purpose

Launches an external file viewer. Typically, you use this method to display test reports.

Remarks

The method chooses the external viewer based on the settings that the user can edit when you call DisplayExternalViewerDialog. If the dialog box contains no settings that apply to the file, TestStand launches a viewer based on the file extension association defined for the operating system.

Parameter	Type	Description
filePath	String	The pathname of the file to view.

See Also

DisplayExternalViewerDialog

NewEditArgs Method

Syntax

Engine.**NewEditArgs**()

Return Type

EditArgs

Purpose

Creates and returns an EditArgs object. Release your reference to this object when you finish using it.

See Also

EditArgs

NewExecution Method

Syntax

Engine.**NewExecution**(sequenceFileParam, sequenceName, processModelParam, breakAtFirstStep, executionTypeMaskParam [, sequenceArgsParam] [, editArgsParam] [, InteractiveArgsParam])

Return Type

Execution

Purpose

Creates and returns a new Execution object. The execution begins immediately.

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

Remarks

Sequence editor and run-time operator interface programs use this method to run sequences.

Parameter	Type	Description
sequenceFileParam	SequenceFile	Pass the sequence file object that contains the sequence to execute. If the execution uses a process model, pass the client sequence file object.
sequenceName	String	Pass the name of the sequence or process model entry point to execute.
processModelParam	SequenceFile	Pass the process model sequence file object if you want to execute a process model entry point. Otherwise, pass a NULL dispatch pointer in MFC, a NULL object reference in LabVIEW, 0 in LabWindows/CVI, or the Nothing keyword in Visual Basic.
breakAtFirstStep	Boolean	Pass True to suspend execution before executing the first step.
executionTypeMaskParam	Long	Pass 0 for the default behavior or pass one or more ExecutionTypeMask constants. Use the bitwise-OR operator to pass multiple constants.
sequenceArgsParam	Variant	Pass a PropertyObject object that contains the arguments to the sequence you want to execute. Each subproperty of the PropertyObject object represents an parameter to the sequence. The subproperties must appear in the same order as the sequence parameters. This parameter is optional.
editArgsParam	Variant	Pass an EditArgs object that indicates which items are currently selected in the operator interface. This is required only for process model entry points.
InteractiveArgsParam	Variant	Pass an InteractiveArgs object that indicates which steps are currently selected in the operator interface, and contains looping information necessary for an interactive execution. Pass this parameter only for interactive executions.

See Also

Execution
EditArgs
InteractiveArgs
ExecutionTypeMask

NewInteractiveArgs Method

Syntax

Engine.**NewInteractiveArgs**()

Return Type

InteractiveArgs

Purpose

Creates and returns an InteractiveArgs object. Release your reference to this object when you finish using it.

See Also

InteractiveArgs

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

NewPropertyObject Method

Syntax

Engine.**NewPropertyObject**(valueType, asArray, typeName, options)

Return Type

PropertyObject

Purpose

Creates and returns a new PropertyObject object. Release your reference to this object when you finish using it.

Parameter	Type	Description
valueType	PropertyValueTypes	Pass the type of value you want the property to contain.
asArray	Boolean	Pass True if you want the property value to be an array.
typeName	String	If you pass PropValType_NamedType for the valueType, pass the name of the type for this parameter. Otherwise, pass an empty string. If you pass a type name, the type must already be in memory. For a list of built-in named types see the NamedPropertyTypes constants in this help file.
options	Long	Pass zero to specify the default behavior, or pass one or more PropertyOptions constants. Use the bitwise-OR operator to specify multiple options.

See Also

PropertyObject
PropertyOptions
NamedPropertyTypes

NewSequence Method

Syntax

Engine.**NewSequence**()

Return Type

Sequence

Purpose

Creates and returns a new Sequence object. Release your reference to this object when you finish using it.

Remarks

Pass the new Sequence object to the SequenceFile.InsertSequence method to add a sequence to a sequence file.

See Also

Sequence

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

NewSequenceFile Method

Syntax

Engine.**NewSequenceFile**()

Return Type

SequenceFile

Purpose

Creates and returns a new SequenceFile object.

Remarks

Call ReleaseSequenceFile on this object before you release it.

See Also

SequenceFile
ReleaseSequenceFile

NewStep Method

Syntax

Engine.**NewStep**(adapterKeyName, stepTypeName)

Return Type

Step

Purpose

Creates and returns a new Step object. Release your reference to this object when you finish using it.

Remarks

Pass the new Step object to the Sequence.InsertStep method to add a step to a sequence.

Parameter	Type	Description
adapterKeyName	String	Pass the key name of the module Adapter object to use to create the step. Pass an empty string to use the adapter that the DefaultAdapter property specifies.
stepTypeName	String	Pass the name of the step type with which to create the step. The step type must already be in memory. For a list of names for the built-in step types, see the StepTypes constants in this help file.

See Also

Step

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

NewUser Method

Syntax

Engine.**NewUser**(userProfile)

Return Type

User

Purpose

Creates and returns a new User object. Release your reference to this object when you finish using it.

Remarks

Typically, only sequence editor programs use this method.

Parameter	Type	Description
userProfile	User	Pass the user profile with which to create the user. Obtain user profiles using the GetUserProfile method.

See Also

User
GetUserProfile

NotifyEndOfModalDialog Method

Syntax

Engine.**NotifyEndOfModalDialog**(modalID)

Purpose

Call this method to notify TestStand that you are no longer displaying the modal dialog box that the modalID parameter specifies.

Remarks

TestStand uses this notification to determine when to reenable your application's main window. Do NOT call this method when displaying a dialog box that the TestStand API implements.

Parameter	Type	Description
modalID	Long	Pass the modalID you receive as a return value from NotifyStartOfModalDialog.

See Also

NotifyStartOfModalDialog

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

NotifyStartOfModalDialog Method

Syntax

Engine.**NotifyStartOfModalDialog**()

Return Type

Long

Purpose

Call this method to notify TestStand before you display a modal dialog box. It returns a modalID which you pass to NotifyEndOfModalDialog when the dialog box is closed.

Remarks

TestStand uses this notification to determine when to disable the main window of your application, to enforce the modality of your dialog box. Do NOT call this method when displaying a dialog box that the TestStand API implements.

Return Value

Returns the modalID to pass to NotifyEndOfModalDialog when your dialog box no longer appears on screen.

See Also

NotifyEndOfModalDialog

RegisterUIMessageCallback Method

Syntax

Engine.**RegisterUIMessageCallback**(callbackFuncAddr)

Purpose

Call this method to register a C function as the UIMessage event callback. You can use a C function as the event callback instead of using ActiveX events or polling. Make sure the UIMessagePollingEnabled property is set to False or your callback will not be called.

Remarks

The callback function must use the following C function prototype:

void __cdecl UIMessageCallback(struct IDispatch *UIMessageDisp);

The UIMessageDisp parameter is the IDispatch pointer to a UIMessage object. Because this pointer is passed to your callback as a parameter, do NOT release it when you finish using it.

Parameter	Type	Description
callbackFuncAddr	Long	Pass the address of the callback function.

See Also

UIMessageDelay
UIMessageMinDelay
UIMessagePollingEnabled
GetUIMessage
UIMessageEvent

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

ReleaseSequenceFile Method

Syntax

Engine.**ReleaseSequenceFile**(sequenceFileToRelease)

Purpose

If you acquire a sequence file from either the GetSequenceFile or NewSequenceFile methods, you must call ReleaseSequenceFile to release the sequence file from the Engine's internal cache before you release your ActiveX reference to it.

Parameter	Type	Description
sequenceFileToRelease	SequenceFile	Pass a reference to the sequence file to release.

See Also

GetSequenceFile
NewSequenceFile

ReloadGlobals Method

Syntax

Engine.**ReloadGlobals**()

Purpose

Reloads global variables from disk.

ShutDown Method

Syntax

Engine.**ShutDown**(final)

Purpose

Attempts to close all open sequence files and terminate all executions. If you are writing an operator interface, call this function before exiting your application.

Remarks

TestStand sends a UIMsg_ShutDownComplete message to notify you when the shut down is complete.

Parameter	Type	Description
final	Boolean	Pass True when performing the final shutdown before exiting the application.

TerminateAll Method

Syntax

Engine.**TerminateAll**()

Purpose

Terminates all existing executions.

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

UIMessageEvent Method

Syntax

Engine.**UIMessageEvent**(msg)

Purpose

Handles UIMessage events as ActiveX events. You must implement this event if you are writing an operator interface and you want to receive UIMessages as ActiveX events. If you are using the TestStand ActiveX control in Visual Basic, double-click on the TestStand control icon on your form to implement the ActiveX event callback.

Parameter	Type	Description
msg	UIMessage	The UIMessage that TestStand passes to your event.

See Also

UIMessage
UIMessageDelay
UIMessageMinDelay
UIMessagePollingEnabled
GetUIMessage
RegisterUIMessageCallback

UnloadAllModules Method

Syntax

Engine.**UnloadAllModules**()

Purpose

Unloads all code modules associated with steps, steptypes, and substeps.

Remarks

Call this function to force all modules to be unloaded and then reloaded when they are next called. This allows you to rebuild DLLs and other code modules.

UserNameExists Method

Syntax

Engine.**UserNameExists**(loginName)

Return Type

Boolean

Purpose

Returns True if a user with a specific login name exists.

Parameter	Type	Description
loginName	String	The login name to search for.

See Also

User

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

Execution

Objects of the Execution class represent executions the user initiates using an operator interface. Use objects of this class to control and get information about executions. For example, you can use methods to suspend, resume, or terminate execution, and you can use properties to determine whether the execution is an interactive execution or whether it is in the process of terminating.

Refer to Chapter 6, *Sequence Execution*, of the *TestStand User Manual* for more information about executions.

Properties

BreakOnRTEForThisExecution
DisplayName
ErrorObject
ForegroundThreadIndex
Id
InInteractiveMode
ModelPostInteractiveCallbackEnabled
ModelPostStepCallbackEnabled
ModelPreInteractiveCallbackEnabled
ModelPreStepCallbackEnabled
NumThreads
Report
ResultObject
ResultStatus
SeqFilePostInteractiveCallbackEnabled
SeqFilePostStepCallbackEnabled
SeqFilePreInteractiveCallbackEnabled
SeqFilePreStepCallbackEnabled
StandardResultsEnabled
StationPostInteractiveCallbackEnabled
StationPostStepCallbackEnabled
StationPreInteractiveCallbackEnabled
StationPreStepCallbackEnabled
TimeResultsEnabled
TypeMask

Methods

Abort
AddExtraResult
AsPropertyObject
Break
CancelTermination
ClearExtraResultList
DeleteExtraResult
GetModelSequenceFile
GetSequenceFile
GetStates
GetThread
Restart
Resume
StepInto
StepOut
StepOver
Terminate
TerminateInteractiveExecution
WaitForEnd

BreakOnRTEForThisExecution Property

Syntax

Execution.**BreakOnRTEForThisExecution**()

Data Type

Boolean

Purpose

Set this property to False if you do not want to receive a break event on run-time errors for the execution.

Remarks

If you set this property to False, the next run-time error causes the execution to terminate. Cleanup step groups of the active sequences run as part of the termination procedure.

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

DisplayName Property (Read Only)

Syntax

Execution.**DisplayName**()

Data Type

String

Purpose

Returns the name to display for the execution.

ErrorObject Property (Read Only)

Syntax

Execution.**ErrorObject**()

Data Type

PropertyObject

Purpose

Returns the error PropertyObject which contains the error reporting properties for this execution. Release your reference to this object when you have finished using it.

ForegroundThreadIndex Property

Syntax

Execution.**ForegroundThreadIndex**()

Data Type

Long

Purpose

Use this property to get and set the foreground thread of the execution.

Remarks

You specify the thread with a zero-based index into the list of threads in the execution. The foreground thread is the thread that the sequence editor or run-time operator interface displays as active for the execution.

See Also

NumThreads

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

Id Property (Read Only)

Syntax

Execution.**Id**()

Data Type

Long

Purpose

Returns a unique ID number for the execution. The ID number is unique with respect to all executions that you might initiate before you shut down the TestStand engine.

Remarks

Use this ID number to compare two Execution object references to determine whether or not they refer to the same underlying execution.

InInteractiveMode Property (Read Only)

Syntax

Execution.**InInteractiveMode**()

Data Type

Boolean

Purpose

Returns True if the execution is executing steps interactively.

Remarks

This property is always True for executions that you create as interactive executions. For executions that you start as normal executions, this property is True only when you execute steps interactively while at a breakpoint state.

ModelPostInteractiveCallbackEnabled Property

Syntax

Execution.**ModelPostInteractiveCallbackEnabled**()

Data Type

Boolean

Purpose

Set this property to False to disable the ProcessModelPostInteractive callback.

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

ModelPostStepCallbackEnabled Property

Syntax

Execution.**ModelPostStepCallbackEnabled**()

Data Type

Boolean

Purpose

Set this property to False to disable the ProcessModelPostStep callback.

ModelPreInteractiveCallbackEnabled Property

Syntax

Execution.**ModelPreInteractiveCallbackEnabled**()

Data Type

Boolean

Purpose

Set this property to False to disable the ProcessModelPreInteractive callback.

ModelPreStepCallbackEnabled Property

Syntax

Execution.**ModelPreStepCallbackEnabled**()

Data Type

Boolean

Purpose

Set this property to False to disable the ProcessModelPreStep callback.

NumThreads Property (Read Only)

Syntax

Execution.**NumThreads**()

Data Type

Long

Purpose

Returns the current number of threads in the execution.

See Also

ForegroundThreadIndex

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

Report Property (Read Only)

Syntax

Execution.**Report**()

Data Type

Report

Purpose

Returns the Report object for the execution.

ResultObject Property (Read Only)

Syntax

Execution.**ResultObject**()

Data Type

PropertyObject

Purpose

Returns the "Result" PropertyObject for the execution. Release your reference to this object when you have finished using it.

Remarks

The "Result" PropertyObject contains the following subproperties:

* **Error** -- a property object that contains the error information for the execution.
* **ResultList** -- an array that contains the combined results for all the steps in the execution.
* **SequenceFile** -- a string that contains the path of the sequence file associated with the execution.
* **Sequence** -- a string that contains the name of the sequence associated with the execution.
* **Status** -- a string that contains the status of the execution.

You also can get the Error subproperty of the "Result" PropertyObject by using the ErrorObject property, and you can access the Status subproperty by using the ResultStatus property.

ResultStatus Property

Syntax

Execution.**ResultStatus**()

Data Type

String

Purpose

Returns the result status string for the execution.

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

SeqFilePostInteractiveCallbackEnabled Property

Syntax

Execution.**SeqFilePostInteractiveCallbackEnabled**()

Data Type

Boolean

Purpose

Set this property to False to disable the SequenceFilePostInteractive callback.

SeqFilePostStepCallbackEnabled Property

Syntax

Execution.**SeqFilePostStepCallbackEnabled**()

Data Type

Boolean

Purpose

Set this property to False to disable the SequenceFilePostStep callback.

SeqFilePreInteractiveCallbackEnabled Property

Syntax

Execution.**SeqFilePreInteractiveCallbackEnabled**()

Data Type

Boolean

Purpose

Set this property to False to disable the SequenceFilePreInteractive callback.

SeqFilePreStepCallbackEnabled Property

Syntax

Execution.**SeqFilePreStepCallbackEnabled**()

Data Type

Boolean

Purpose

Set this property to False to disable the SequenceFilePreStep callback.

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

StandardResultsEnabled Property

Syntax

Execution.**StandardResultsEnabled**()

Data Type

Boolean

Purpose

Set this property to False to disable storing of all the standard results for the execution.

Remarks

The "standard results" are the subproperties in the TS subproperty of the Results property for each step. For more information, refer to the Result Collection section in Chapter 6, *Sequence Execution*, of the *TestStand User Manual*.

StationPostInteractiveCallbackEnabled Property

Syntax

Execution.**StationPostInteractiveCallbackEnabled**()

Data Type

Boolean

Purpose

Set this property to False to disable the StationPostInteractive callback.

StationPostStepCallbackEnabled Property

Syntax

Execution.**StationPostStepCallbackEnabled**()

Data Type

Boolean

Purpose

Set this property to False to disable the StationPostStep callback.

StationPreInteractiveCallbackEnabled Property

Syntax

Execution.**StationPreInteractiveCallbackEnabled**()

Data Type

Boolean

Purpose

Set this property to False to disable the StationPreInteractive callback.

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

StationPreStepCallbackEnabled Property

Syntax

Execution.**StationPreStepCallbackEnabled**()

Data Type

Boolean

Purpose

Set this property to False to disable the StationPreStep callback.

TimeResultsEnabled Property

Syntax

Execution.**TimeResultsEnabled**()

Data Type

Boolean

Purpose

Set this property to False to disable the storing of timing results.

TypeMask Property (Read Only)

Syntax

Execution.**TypeMask**()

Data Type

Long

Purpose

Returns the ExecutionTypeMask constants with which you created the execution.

See Also

ExecutionTypeMask

Abort Method

Syntax

Execution.**Abort**()

Purpose

Call this method to abort the execution. Cleanup step groups do not execute as part of the abort process.

Remarks

The execution might not terminate immediately. If a call to a step module is active, the execution waits for that step module to return.

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

AddExtraResult Method

Syntax

Execution.**AddExtraResult**(propertyName, resultPropertyName)

Purpose

Call this method to specify a step property for TestStand to copy to the results list after the execution of each step.

Remarks

TestStand always copies each subproperty within the Result property of each step to the result list, but only if the step has a Result property. You can use this function to specify an additional step property to include in the result list. If a step does not have the property you specify, it will not add that property to the result list element for that step. You can add any number of extra results by calling this method repeatedly.

Parameter	Type	Description
propertyName	String	The property name of the property to add to the results. For example, to add the upper limit of numeric measurement steps, you pass the name "Step.Limits.High".
resultPropertyName	String	The name with which to store the copy of the property in the results list entry for each step.

See Also

ClearExtraResultList
DeleteExtraResult

AsPropertyObject Method

Syntax

Execution.**AsPropertyObject**()

Return Type

PropertyObject

Purpose

Returns the underlying PropertyObject that represents the Execution object. Use the PropertyObject to modify, add, or remove custom properties of the object.

Break Method

Syntax

Execution.**Break**()

Purpose

Call this method to suspend the execution.

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

CancelTermination Method

Syntax

Execution.**CancelTermination**()

Purpose

Call this method to cancel the termination of an execution that is in the process of terminating.

Remarks

Call this method from within a step. Calling this method from the main thread of the operator interface or from within an edit substep of a step type will result in deadlock.

ClearExtraResultList Method

Syntax

Execution.**ClearExtraResultList**()

Purpose

Clears the list of properties to add to the results list.

See Also

AddExtraResult
DeleteExtraResult

DeleteExtraResult Method

Syntax

Execution.**DeleteExtraResult**(propertyName)

Purpose

Removes a specific property from the list of properties to add to the results.

Parameter	Type	Description
propertyName	String	The property name of the property to remove from the list of properties to add to the results. Use the same name that you passed as the first parameter to the AddExtraResult method. For example: "Step.Limits.High".

See Also

AddExtraResult
ClearExtraResultList

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

GetModelSequenceFile Method

Syntax

Execution.**GetModelSequenceFile**()

Return Type

SequenceFile

Purpose

Returns the sequence file object for the process model file the execution is using.

Remarks

Returns a NULL object reference if the execution is not using a process model. Release your reference to this object when you have finished using it.

GetSequenceFile Method

Syntax

Execution.**GetSequenceFile**()

Return Type

SequenceFile

Purpose

Returns the sequence file object that contains the sequence in which the execution began.

Remarks

If the execution is using a process model, the method returns the sequence file object for the client sequence file. Release your reference to this object when you have finished using it.

GetStates Method

Syntax

Execution.**GetStates**(runState, termState)

Purpose

Returns the current state of the execution.

Parameter	Type	Description
runState	ExecutionRunStates	
termState	ExecutionTerminationStates	

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

GetThread Method

Syntax

Execution.**GetThread**(index)

Return Type

Thread

Purpose

Returns a thread object for the thread that you identify by index.

Remarks

Release your reference to this object when you have finished using it.

Parameter	Type	Description
index	Long	A zero-based index into the list of threads in the execution.

See Also

NumThreads
ForegroundThreadIndex

Restart Method

Syntax

Execution.**Restart**(breakOnEntry)

Purpose

Restarts a completed execution.

Remarks

Execution restarts from the beginning.

Parameter	Type	Description
breakOnEntry	Boolean	

Resume Method

Syntax

Execution.**Resume**()

Purpose

Call this method to resume the execution from a suspended state.

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

StepInto Method

Syntax

Execution.**StepInto**()

Purpose

Call this method to resume the execution from a suspended state and to suspend again at the earliest possible point.

Remarks

If the step module Adapter for the next step allows you to debug into the step module, execution suspends at the beginning of the step module code. If the step is a sequence call step, execution suspends at the first step in the subsequence.

StepOut Method

Syntax

Execution.**StepOut**()

Purpose

Call this method to resume the execution from a suspended state and to suspend again after execution of the current sequence completes.

StepOver Method

Syntax

Execution.**StepOver**()

Purpose

Call this method to resume the execution from a suspended state and to suspend again after execution of the next step completes.

Terminate Method

Syntax

Execution.**Terminate**()

Purpose

Call this method to terminate the execution. Cleanup step groups execute as part of the termination process. If the execution is an interactive execution, use TerminateInteractiveExecution instead.

Remarks

The execution might not terminate immediately. If a call to a step module is active, the execution waits for that step module to return.

See Also

TerminateInteractiveExecution

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

TerminateInteractiveExecution Method

Syntax

Execution.**TerminateInteractiveExecution**()

Purpose

Call this method to terminate an interactive execution. If you started the interactive execution while suspended in a normal execution, execution suspends at the point in the original execution at which you started the interactive execution.

Remarks

The Cleanup step group for the sequence in which you started the interactive execution runs only if you created the execution as an interactive execution and the user enables the Run Setup and Cleanup for Interactive Execution option in the Station Options dialog box.

The execution might not terminate immediately. If a call to a step module is active, the execution waits for that step module to return.

See Also

Terminate

WaitForEnd Method

Syntax

Execution.**WaitForEnd**(millisecondTimeOut, processWindowsMsgs = True [, callingExecution])

Return Type

Boolean

Purpose

Use this function to wait for an execution to end.

Remarks

Not meant to be used by an operator interface or sequence editor as it does not process UIMessages. Instead use this method from a step to synchronize with another execution.

Return Value

Returns True when the execution ends, or False when the timeout occurs.

Parameter	Type	Description
millisecondTimeOut	Long	Specify the maximum number of milliseconds to wait. Specify -1 for no timeout.
processWindowsMsgs	Boolean	Specify True to process Windows messages while waiting, or False to wait without processing Windows messages. This parameter has a default value of True.
callingExecution	Variant	This is an optional parameter. If you call this method from a step, pass a reference to the step's execution in order to allow this method to return immediately when the step's execution is terminated or aborted.

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

ExecutionMask Constants

Use these constants to specify the Engine.ExecutionMask property.

Each constant corresponds to an execution option. The user typically sets these options in the Execution tab of the Station Options dialog box.

- ExecMask_BreakOnRunTimeError
- ExecMask_BreakpointsEnabled
- ExecMask_BreakWhileTerminating
- ExecMask_DefaultExecutionMask
- ExecMask_TraceAllThreads
- ExecMask_TraceIntoEntryPoints
- ExecMask_TraceIntoPostActionCallbacks
- ExecMask_TraceIntoPrePostCallbacks
- ExecMask_TraceIntoSeparateExecutionCallbacks
- ExecMask_TraceIntoSequenceCallsMarkedAsTraceOff
- ExecMask_TraceIntoSetupCleanup
- ExecMask_TraceWhileTerminating
- ExecMask_TracingEnabled

ExecutionRunStates Enumeration

You can use the following constants with this data type.

- ExecRunState_Paused
- ExecRunState_Running
- ExecRunState_Stopped

ExecutionTerminationStates Enumeration

You can use the following constants with this data type.

- ExecTermState_Aborting
- ExecTermState_KillingThreads
- ExecTermState_Normal
- ExecTermState_Terminating
- ExecTermState_TerminatingInteractive

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

ExecutionTypeMask Constants

Use these constants to specify the executionTypeMaskParam parameter of the Engine.NewExecution method.

- **ExecTypeMask_InitiallyHidden** -- Specifies that an execution does not appear in a window unless a trace or break event occurs. Typically, you use this for exections that you do not want the user to see unless an error occurs.
- **ExecTypeMask_Normal** -- Execution appears in a window when it starts.

FindFilePromptOptions Enumeration

Use these constants to specify the promptOption parameter of the Engine.FindFile method.

The promptOption parameter specifies whether to prompt the user for a file location when TestStand cannot find the file in the search paths.

You can use the following constants with this data type.

- **FindFile_PromptDisable** -- Do not prompt the user to locate the file.
- **FindFile_PromptEnable** -- Prompt the user to locate the file.
- **FindFile_PromptHonorUserPreference** -- Prompt the user to locate the file if the Prompt to Find Files option in the Preferences tab of the Station Options dialog box is enabled.

FindFileSearchListOptions Enumeration

Use these constants to specify the srchListOption parameter of the Engine.FindFile method.

The srchListOption specifies whether TestStand adds the directory of the file that the user selects to the search paths.

You can use the following constants with this data type.

- **FindFile_AddDirToSrchList_Ask** -- Prompt the user to add the directory to the search paths.
- **FindFile_AddDirToSrchList_No** -- Do not add the directory to the search paths.
- **FindFile_AddDirToSrchList_Yes** -- Add the directory to the search paths.

FindPathStatusValues Enumeration

The Engine.FindPath method returns one of these constants to indicate the results of the path search.

You can use the following constants with this data type.

- **FindPath_PathIsDir** -- The FindPath method returned the path of a directory.
- **FindPath_PathIsFile** -- The FindPath method returned the path of a file.
- **FindPath_PathNotFound** -- The FindPath method failed to locate a directory or file with the name you specified.
- **FindPath_PathNotValid** -- The pathname you passed to the FindPath method is not vaild.

FlexCStepAdditions Constants

Use these string constants to create lookupStrings to access the Flexible DLL adapter-specific properties of a step. Notice that the FlexCStep_ExternalCallProp constant refers to the property that contains all the properties for which the constants begin with the ExternalCall_ prefix. The ExternalCall_ParametersProp constant represents the property that contains an array of objects that describe the return value and parameters for the function. The constants having names that begin with FCParam_ apply to the return value and each parameter.

- **ExternalCall_CallConvProp** -- A number property that identifies the calling convention of the DLL function. 0 = cdecl = 0. 1= stdcall.
- **ExternalCall_FunctionNameProp** -- A string that contains the name of the C function to call.
- **ExternalCall_LibPathProp** -- A string that contains the DLL path.

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

- **ExternalCall_ParametersProp** -- An array of objects that describe the parameters to the function. The first element describes the return value of the function. Each element is of type FCParameter.
- **FCParam_ArgValueProp** -- A string property that contains the argument expression for the parameter.
- **FCParam_FlagsProp** -- A number property that can contain the following bit flags: SetErrorToReturnValue = (1L << 1)
- **FCParam_NameProp** -- A string property that contains the name of the parameter. The name is for display purposes only.
- **FCParam_NumArrayElementsProp** -- A string property containing the expression for the number of elements to pass. The property applies only if the parameter is an array or string buffer.
- **FCParam_NumericPassingProp** -- A number property that specifies how to pass a numeric parameter. 0 = PassValue. 1 = PassPointer.
- **FCParam_NumericTypeProp** -- A number that indicates the particular data type of a numeric parameter. 0 = signed 8-bit integer. 1 = unsigned 8-bit integer. 2 = signed 16-bit integer. 3 = unsigned 16-bit integer. 4 = signed 32-bit integer. 5 = unsigned 32-bit integer. 6 = 32-bit floating point number. 7 = 64-bit floating point number.
- **FCParam_ObjectTypeProp** -- A number property that specifies the type of data to pass to represent an object parameter. 0 = IDispatchPtr. 1 = CVIAutomationHandle. 2 = IUnknownPtr.
- **FCParam_ResultActionProp** -- A number property that indicates whether and how the output value from the parameter can trigger a run-time error. 0 = no action. 1 = run-time error if value is less than zero. 2 = run-time error if value is greater than zero. 3 = run-time error if value is equal to zero. 4 = run-time error if value is less nonzero.
- **FCParam_StringPassingProp** -- A number property that specifies how to pass a string parameter. 0 = C-style string. 1 = UniCode string. 2 = C-style string buffer.
- **FCParam_TypeProp** -- A number property that specifies the parameter type. 0 = mumeric. 1 = array. 2 = string. 3 = void, 4 = object. Only the return value can use 'void'.
- **FlexCStep_ExternalCallProp** -- The subproperty of the TSInfo_StepAdditions property that contains these sub-properties: ExternalCall_CallConvProp, ExternalCall_FunctionNameProp, ExternalCall_LibPathProp, and ExternalCall_ParametersProp.

GetSeqFileFlags Constants

Use these constants to modify the behavior of the Engine.GetSequenceFile method.

- **GetSeqFile_AllowTypeConflicts** -- If this flag is set, conflicts that TestStand encounters between the sequence and types currently in memory do not prevent it from loading the sequence. If this flag is not set, then TestStand returns an error as soon as it encounters a type conflict.
- **GetSeqFile_CheckModelOptions** -- If this flag is set, TestStand verifies that the sequence file does not refer to a specific process model file that conflicts with the current settings in the Model tab of the Station Options dialog box.
- **GetSeqFile_OperatorInterfaceFlags** -- The standard set of flags that an operator interface program uses. This is the combination of the GetSeqFile_CheckModelOptions, GetSeqFile_PreloadModules, and GetSeqFile_UpdateFromDisk flags.
- **GetSeqFile_PreloadModules** -- If this flag is set, TestStand honors the Module Load Options for the sequence. Otherwise, it does not preload any of the modules used by the steps in the sequence.
- **GetSeqFile_UpdateFromDisk** -- If this flag is set and the sequence is currently in memory, TestStand checks the date of the file on disk and reloads it if it is newer. If the flag is not set and the sequence is currently in memory, TestStand does not load the file.

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

InteractiveArgs

When you call the Engine.NewExecution, or Thread.DoInteractiveExecution method to create an interactive execution, use an object of this class to pass information about the current state of the operator interface. These methods use the object to determine which steps are currently selected in the operator interface.

Create objects of this class using Engine.NewInteractiveArgs.

Properties

LoopCount	NumSteps	StepGroup
StopExpression		

Methods

AddStepIndex	AsPropertyObject	ClearStepList
ContainsStep	GetStepIndex	

See Also

Engine.NewInteractiveArgs
Engine.NewExecution
Thread.DoInteractiveExecution

LoopCount Property

Syntax

InteractiveArgs.**LoopCount**()

Data Type

Long

Purpose

Use this property to get or set the number of times to execute the list of selected steps.

Remarks

If the value of this property is -1, the execution loops indefinitely on the selected steps.

NumSteps Property (Read Only)

Syntax

InteractiveArgs.**NumSteps**()

Data Type

Long

Purpose

Returns the number of step indices added to the InteractiveArgs object.

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

StepGroup Property

Syntax

InteractiveArgs.**StepGroup**()

Data Type

StepGroups

You can use the following constants with this data type.

- StepGroup_Cleanup
- StepGroup_Main
- StepGroup_Setup

Purpose

Use this property to get or set the step group containing the steps to execute interactively.

StopExpression Property

Syntax

InteractiveArgs.**StopExpression**()

Data Type

String

Purpose

Use this property to get or set an expression whose value indicates when to stop the execution.

Remarks

If the expression evaluates to True, the interactive execution stops. If the expression is an empty string then it is not evaluated.

AddStepIndex Method

Syntax

InteractiveArgs.**AddStepIndex**(stepIndexParam)

Purpose

Adds a step to the list of steps to execute interactively

Parameter	Type	Description
stepIndexParam	Long	Pass a zero-based index specifying a step in the list of steps for the sequence in which you are running steps interactively.

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

AsPropertyObject Method

Syntax

InteractiveArgs.**AsPropertyObject**()

Return Type

PropertyObject

Purpose

Returns the underlying PropertyObject that represents the InteractiveArgs object. Use the PropertyObject to modify, add, or remove your own custom properties of the object.

Remarks

Do not use this function to remove or modify the dynamic properties that TestStand already provides for InteractiveArgs objects. You can do this through the other methods of the EditArgs class.

ClearStepList Method

Syntax

InteractiveArgs.**ClearStepList**()

Purpose

Clears the list of steps to execute interactively.

ContainsStep Method

Syntax

InteractiveArgs.**ContainsStep**(stepIndexParam)

Return Type

Boolean

Purpose

Returns True if the step you specify is in the list of steps to run interactively.

Parameter	Type	Description
stepIndexParam	Long	Pass a zero-based index specifying a step in the list of steps for the sequence in which you are running steps interactively.

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

GetStepIndex Method

Syntax

InteractiveArgs.**GetStepIndex**(arrayIndexParam)

Return Type

Long

Purpose

Returns a step index stored at a specific position in the list of steps to run interactively.

Remarks

The step index that the method returns is a zero-based index into the list of steps for the sequence.

Parameter	Type	Description
arrayIndexParam	Long	Pass a zero-based index into the list of steps to run interactively.

InteractiveContext

Use objects of this class to obtain additional information about a currently executing interactive execution. Get objects of this class using the SequenceContext.InteractiveContext property of the sequence context in which the interactive execution began.

Properties

IsRootExecution SavedNextStepIndex SavedPreviousStepIndex SavedStepIndex

Methods

AsPropertyObject

See Also

SequenceContext.InteractiveContext

IsRootExecution Property (Read Only)

Syntax

InteractiveContext.**IsRootExecution**()

Data Type

Boolean

Purpose

Returns True if the execution is a root interactive execution. A root interactive execution is a new, independent execution that the user initiates on selected steps from a sequence file window.

Returns False if the execution is a nested interactive execution. The user starts a nested interactive execution from an execution window for a normal execution that is suspended at a breakpoint. The nested interactive execution runs within the context of the normal execution.

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

SavedNextStepIndex Property (Read Only)

Syntax

InteractiveContext.**SavedNextStepIndex**()

Data Type

Long

Purpose

This property applies only to nested interactive executions. It returns the index of the step that was ready to execute next in the normal execution when the user began the current interactive execution. In other words, if the IsRootExecution property is False, this property returns the value of the SequenceContext.NextStepIndex property at the time the interactive execution began. This value is purely informational. It is provided so that you can display such information to the user when writing an operator interface.

Remarks

Returns -1 if the IsRootExecution property is True.

SavedPreviousStepIndex Property (Read Only)

Syntax

InteractiveContext.**SavedPreviousStepIndex**()

Data Type

Long

Purpose

This property applies only to nested interactive executions. It returns the index of the last step that executed in the normal execution before the user began the current interactive execution. In other words, if the IsRootExecution property is False, this property returns the value of the SequenceContext.PreviousStepIndex property at the time the interactive execution began. This value is purely informational. It is provided so that you can display such information to the user when writing an operator interface.

Remarks

Returns -1 if the IsRootExecution property is True.

SavedStepIndex Property (Read Only)

Syntax

InteractiveContext.**SavedStepIndex**()

Data Type

Long

Purpose

This property applies only to nested interactive executions. It returns the index of the step that was active in the normal execution when the user began the current interactive execution. In other words, if the IsRootExecution property is False, this property returns the state of the SequenceContext.StepIndex property before the interactive execution began.

Currently, because you can only begin an interactive execution between steps in the currently executing sequence, this property always returns -1.

Remarks

Returns -1 if the IsRootExecution property is True.

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

AsPropertyObject Method

Syntax

InteractiveContext.**AsPropertyObject**()

Return Type

PropertyObject

Purpose

Returns the underlying PropertyObject that represents the InteractiveContext object. Use the PropertyObject to modify, add, or remove custom properties of the object.

LVStepAdditions Constants

Use these string constants to create lookupStrings to access the adapter-specific properties of a LabVIEW step. The following example code obtains the module pathname of a LabVIEW step:

TS_PropertySetValString(propObj, &errorInfo, Step_TSInfoProp "." TSInfo_StepAdditions "." LVStep_ModulePathProp, 0, testFilePath);

- **LVStep_ModulePathProp** -- Pathname of the VI called by the step.
- **LVStep_PassInBufProp** -- A Boolean property that controls whether TestStand passes the Input Buffer parameter to the VI.
- **LVStep_PassInvocInfoProp** -- A Boolean property that controls whether TestStand passes the Invocation Info parameter to the VI.
- **LVStep_PassSeqContextProp** -- A Boolean property that controls whether TestStand passes an ActiveX pointer to sequence context as a parameter to the VI.
- **LVStep_ShowFrontPanelProp** -- A Boolean property that controls whether the adapter activates the front panel of the VI when the step calls the VI.

ModuleLoadOptions Enumeration

Use these constants to set the sequence file and step code module options that control when TestStand loads code modules and subsequence files. When you use one of these constants with the SequenceFile.ModuleLoadOption property, it applies to all steps in the sequence file. When you use one of these constants with the Step.ModuleLoadOption property, it applies to a particular step.

You can use the following constants with this data type.

- **LoadOption_DynamicLoad** -- Do not load the code module for a step until the step is ready to call it.
- **LoadOption_PreloadWhenExecuted** -- Load the code module for a step when any sequence in the sequence file containing the step begins executing.
- **LoadOption_PreloadWhenOpened** -- Load the code module for a step when TestStand loads into memory the sequence file containing the step.
- **LoadOption_UseStepLoadOption** -- Load each code module according to the load option for the step that uses it. This option is valid only for the SequenceFile.ModuleLoadOption property.

See Also

SequenceFile.ModuleLoadOption
Step.ModuleLoadOption

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

ModuleUnloadOptions Enumeration

Use these constants to set the sequence file and step code module options that control when TestStand unloads code modules and subsequence files. When you use one of these constants with the SequenceFile.ModuleUnloadOption property, it applies to all steps in the sequence file. When you use one of these constants with the Step.ModuleUnloadOption property, it applies to a particular step.

You can use the following constants with this data type.

- **UnloadOption_AfterSequenceExecution** -- Unload the code module for a step after the sequence containing the step finishes executing.
- **UnloadOption_AfterStepExecution** -- Unload the code module for a step after the step finishes executing.
- **UnloadOption_OnPreconditionFailure** -- Unload the code module for a step if the precondition for the step evaluates to False.
- **UnloadOption_UseStepUnloadOption** -- Unload the code module for a step according to the unload option for the step. This option is valid only for the SequenceFile.ModuleUnloadOption property.
- **UnloadOption_WithSequenceFile** -- Unload the code module for a step when TestStand unloads from memory the sequence file containing the step.

See Also

SequenceFile.ModuleUnloadOption
Step.ModuleUnloadOption

NamedPropertyTypes Constants

Use these constants to specify one of the built-in, named data types when calling a method such as Engine.NewPropertyObject, PropertyObject.NewSubProperty, or PropertyObject.GetType.

- PropType_CommonResults
- PropType_Error
- PropType_FCParam
- PropType_Path

See Also

Engine.NewPropertyObject
PropertyObject.NewSubProperty
PropertyObject.GetType

OpenFileDialogFlags Constants

Use these constants with the Engine.DisplayOpenFileDialog method to specify options for the file dialog box. The OpenFile dialog box has a checkbox at the bottom with the label "Use Absolute Path". These flags control how that checkbox appears. If you specify no flags, then the checkbox is not dimmed, and a checkmark appears in the checkbox initially if the initialPath parameter to the function is an absolute path.

- **OpenFile_DisableUseAbsPathCheck** -- This flag dims the "Use Absolute Path" checkbox.
- **OpenFile_InitialSetUseAbsPathCheck** -- This flag sets the initial state of the "Use Absolute Path" checkbox to checked when the dialog box first opens.
- **OpenFile_InitialUnsetUseAbsPathCheck** -- This flag sets the initial state of the "Use Absolute Path" checkbox to unchecked when the dialog box first opens.
- **OpenFile_UseAbsolutePath** -- This is the combination of the OpenFile_DisableUseAbsPathCheck and OpenFile_InitialSetUseAbsPathCheck flags. Use this flag when you want to ensure that the method returns only absolute paths.

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

- **OpenFile_UseRelativePath** -- This is the combination of the OpenFile_DisableUseAbsPathCheck and OpenFile_InitialUnsetUseAbsPathCheck flags. Use this flag when you want to ensure that the method returns only relative paths or simple filenames.

See Also

Engine.DisplayOpenFileDialog

PostActionValues Constants

Use these constants to specify the type of action you want to take for a particular post action of a step.

- PostAction_Break
- PostAction_CallCallback
- PostAction_GotoStep
- PostAction_NextStep
- PostAction_Terminate

See Also

Step.FailAction
Step.PassAction
Step.CustomFalseAction
Step.CustomTrueAction

PropertyFlags Constants

These constants represent the flags you can set and get on PropertyObject objects. Use the bitwise-OR operator to specify more than one flag for a particular PropertyObject object.

- **PropFlags_DontTypeCheckParameter** -- Set this flag in a sequence parameter to disable type-checking for that parameter.
- **PropFlags_ExcludeFromComparison** -- Set this flag in a property to ignore the property when its containing object is compared to another object.
- **PropFlags_ExcludeFromCopy** -- Set this flag in a property to prevent the property from being copied when its containing object is copied.
- **PropFlags_Hidden** -- Set this flag in an object to prevent it from appearing in the sequence editor unless the "Show Hidden Properties" station option is set.
- **PropFlags_HiddenInTypes** -- Set this flag in an object to prevent it from being seen in the sequence editor Type View unless the "Show Hidden Properties" station option is set.
- **PropFlags_IsType** -- Indicates that an object is a type or property of a type.
- **PropFlags_NotDeletable** -- Set this flag in any object to prevent the user from deleting the object in a sequence editor.
- **PropFlags_NotEditable** -- Set this option to prevent the user from editing the object or deleting its properties in a sequence editor.
- **PropFlags_PassByReference** -- This flag allows a sequence to control how TestStand passes each parameter to it. Set this flag to pass the parameter by reference. Remove this flag to pass a copy of the parameter.
- **PropFlags_Shared** -- This flag allows you to specify that a property be shared among multiple copies of its containing object. If this flag is set when the containing object is copied, the copy of the object receives a pointer to the property rather than a copy of the property.
- **PropFlags_SharedAtRunTime** -- Similar to PropFlags_Shared, except that the flag is honored only during run-time.
- **PropFlags_UnstructuredProperty** -- Indicates that no type restrictions exist for the property. Separate instances of the property can contain objects of different types.

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

See Also

PropertyObject.GetFlags
PropertyObject.SetFlags

PropertyObject

You use the PropertyObject class to manipulate and access the values of variables and custom step properties. Typically, you use the PropertyObject class in test modules to get or set the values of custom step properties, sequence local variables, sequence file globals, and station global variables. Also, you can use the PropertyObject to add, copy, or delete sub-properties of variables and custom step properties. For most test modules that use the TestStand API, the PropertyObject class is the only TestStand API class that is necessary.

You can create new objects of this class using Engine.NewPropertyObject. You can get existing objects of this class as return values from numerous functions. You can get the underlying PropertyObject for objects of every class except the Engine class by using the AsPropertyObject method of the respective class.

Properties

Comment	Name

Methods

Clone	DeleteSubProperty	Evaluate
Exists	GetArrayIndex	GetArrayOffset
GetDimensions	GetFlags	GetNthSubPropertyName
GetNumSubProperties	GetPropertyObject	GetType
GetTypeDefinition	GetValBoolean	GetValIDispatch
GetValInterface	GetValNumber	GetValString
GetValVariant	NewSubProperty	Read
Serialize	SetDimensions	SetFlags
SetNthSubPropertyName	SetPropertyObject	SetValBoolean
SetValIDispatch	SetValInterface	SetValNumber
SetValString	SetValVariant	Unserialize
Write		

Comment Property

Syntax

PropertyObject.**Comment**()

Data Type

String

Purpose

The comment string associated with the PropertyObject.

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

Name Property

Syntax

PropertyObject.**Name**()

Data Type

String

Purpose

Name of the PropertyObject.

Example

```
LabWindows/CVI:
       char *oldName = NULL;

       TS_PropertyGetProperty(propObj, &errorInfo , TS_PropertyName,
                              CAVT_CSTRING, &oldName);
       TS_PropertySetProperty(propObj, &errorInfo, TS_PropertyName,
                              CAVT_CSTRING, "NewObjName");
       ...
       CA_FreeMemory(oldName);

Visual Basic:
       Dim oldName As String

       oldName = propObj.Name
       propObjName.Name = "NewObjName"

MFC:
       CString oldName;

       oldName = propObj.GetName();
       propObj.SetName("NewObjName");
```

Clone Method

Syntax

PropertyObject.**Clone**(lookupString, options)

Return Type

PropertyObject

Purpose

Creates a copy of the property that lookupString specifies.

Return Value

Copy of the property. Release the copy when you finish using it.

Parameter	Type	Description
lookupString	String	Pass an empty string to denote the PropertyObject to which the method applies, or pass the name of a subproperty within the PropertyObject. See lookupString for more details.
options	Long	Pass 0 to specify the default behavior, or pass one or more PropertyOptions constants. Use the bitwise-OR operator to specify multiple options.

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

See Also
PropertyOptions

DeleteSubProperty Method

Syntax
PropertyObject.**DeleteSubProperty**(lookupString, options)

Purpose
Deletes the subproperty with the name that lookupString specifies.

Parameter	Type	Description
lookupString	String	Pass the name of a subproperty within the PropertyObject. See lookupString for more details.
options	Long	Pass 0 to specify the default behavior, or pass one or more PropertyOptions constants. Use the bitwise-OR operator to specify multiple options.

Evaluate Method

Syntax
PropertyObject.**Evaluate**(exprString)

Return Type
PropertyObject

Purpose
Evaluates an expression and returns the result.

Return Value
Result of the expression, in the form of a PropertyObject. The PropertyObject can contain a value of any type depending on the expression. Release it when you finish using it.

Parameter	Type	Description
exprString	String	Standard TestStand expression

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

Exists Method

Syntax

PropertyObject.**Exists**(lookupString, options)

Return Type

Boolean

Purpose

Returns True if the property that lookupString specifies exists.

Return Value

True if the property exists.

Parameter	Type	Description
lookupString	String	Pass an empty string to denote the PropertyObject to which the method applies, or pass the name of a subproperty within the PropertyObject. See lookupString for more details.
options	Long	Pass 0 to specify the default behavior, or pass one or more PropertyOptions constants. Use the bitwise-OR operator to specify multiple options.

GetArrayIndex Method

Syntax

PropertyObject.**GetArrayIndex**(lookupString, options, offset)

Return Type

String

Purpose

Returns the index of the array element specified by the offset parameter.

Return Value

An array index string. Array index strings are a list of numbers enclosed in brackets that index each dimension of the array. For example, the following is an array index for a two-dimensional array: "[0][1]"

Parameter	Type	Description
lookupString	String	Pass an empty string to denote the PropertyObject to which the method applies, or pass the name of a subproperty within the PropertyObject. See lookupString for more details.
options	Long	Pass 0 to specify the default behavior, or pass one or more PropertyOptions constants. Use the bitwise-OR operator to specify multiple options.
offset	Long	A zero-based number representing the offset of an array element in the one-dimensional physical storage of the array. The offset is in terms of the number of elements.

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

GetArrayOffset Method

Syntax

PropertyObject.**GetArrayOffset**(lookupString, options, arrayIndex)

Return Type

Long

Purpose

Returns the zero-based offset of the array element that the arrayIndex parameter specifies.

Return Value

A zero-based number representing the offset of an array element in the one-dimensional physical storage of the array. The offset is in terms of number of elements.

Parameter	Type	Description
lookupString	String	Pass an empty string to denote the PropertyObject to which the method applies, or pass the name of a subproperty within the PropertyObject. See lookupString for more details.
options	Long	Pass 0 to specify the default behavior, or pass one or more PropertyOptions constants. Use the bitwise-OR operator to specify multiple options.
arrayIndex	String	An array index string. Array index strings are a list of numbers enclosed in brackets that index each dimension of the array. For example, the following is an array index for a two-dimensional array: "[0][1]"

GetDimensions Method

Syntax

PropertyObject.**GetDimensions**(lookupString, options, lowerBounds, upperBounds, numElements, elementType)

Purpose

Returns the upper bounds, lower bounds, total number of elements, and element type for the array specified by the lookupString parameter.

Parameter	Type	Description
lookupString	String	Pass an empty string to denote the PropertyObject to which the method applies, or pass the name of a subproperty within the PropertyObject. See lookupString for more details.
options	Long	Pass 0 to specify the default behavior, or pass one or more PropertyOptions constants. Use the bitwise-OR operator to specify multiple options.
lowerBounds	String	An array index string describing the lowest valid index for each dimension. For example, a two-dimensional array with a lower bound of 0 for the first dimension and 2 for the second would have the lower bound string "[0][2]".
upperBounds	String	An array index string describing the highest valid index for each dimension. For example, a two-dimensional array with an upper bound of 2 for the first dimension and 4 for the second would have the upper bound string "[2][4]".
numElements	Long	Total number of elements stored in the array.
elementType	PropertyValueTypes	Type of elements stored in the array.

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

GetFlags Method

Syntax

PropertyObject.**GetFlags**(lookupString, options)

Return Type

Long

Purpose

Returns the flags setting of the property specified by lookupString.

Return Value

Current flags of the property. Refer to PropertyFlags for possible values.

Parameter	Type	Description
lookupString	String	Pass an empty string to denote the PropertyObject to which the method applies, or pass the name of a subproperty within the PropertyObject. See lookupString for more details.
options	Long	Pass 0 to specify the default behavior, or pass one or more PropertyOptions constants. Use the bitwise-OR operator to specify multiple options.

GetNthSubPropertyName Method

Syntax

PropertyObject.**GetNthSubPropertyName**(lookupString, index, options)

Return Type

String

Purpose

Returns the name of a subproperty within the property specified by lookupString. You identify the property using a zero-based index.

Return Value

Name of the subproperty the index specifies.

Parameter	Type	Description
lookupString	String	Pass an empty string to denote the PropertyObject to which the method applies, or pass the name of a subproperty within the PropertyObject. See lookupString for more details.
index	Long	Pass the index of the subproperty. This index is zero-based.
options	Long	Pass 0 to specify the default behavior, or pass one or more PropertyOptions constants. Use the bitwise-OR operator to specify multiple options.

See Also

GetNumSubproperties

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

GetNumSubProperties Method

Syntax

PropertyObject.**GetNumSubProperties**(lookupString)

Return Type

Long

Purpose

Returns the number of subproperties directly within the property specified by lookupString.

Return Value

Number of subproperties within the property specified by lookupString.

Parameter	Type	Description
lookupString	String	Pass an empty string to denote the PropertyObject to which the method applies, or pass the name of a subproperty within the PropertyObject. See lookupString for more details.

See Also

GetNthSubPropertyName

GetPropertyObject Method

Syntax

PropertyObject.**GetPropertyObject**(lookupString, options)

Return Type

PropertyObject

Purpose

Returns the PropertyObject value of the property specified by the lookupString parameter.

Return Value

PropertyObject value of the property.

Parameter	Type	Description
lookupString	String	Pass an empty string to denote the PropertyObject to which the method applies, or pass the name of a subproperty within the PropertyObject. See lookupString for more details.
options	Long	Pass 0 to specify the default behavior, or pass one or more PropertyOptions constants. Use the bitwise-OR operator to specify multiple options.

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

Example

```
LabWindows/CVI:
       CAObjHandle limits = 0;

       TS_PropertyGetPropertyObject(propObj, &errorInfo, "Step.Limits",
                                    0, &limits);
       ...
       CA_DiscardObjHandle(limits);

Visual Basic:
       Dim limits As PropertyObject

       Set limits = propObj.GetPropertyObject("Step.Limits", 0)

MFC:
       PropertyObject limits;

       limits.AttachDispatch(propObj.GetPropertyObject("Step.Limits", 0));
       // Note that the limits object will be released when the
       // limits variable goes out of scope or is deleted
```

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

GetType Method

Syntax

PropertyObject.**GetType**(lookupString, options, isObject, isArray, typeName)

Return Type

PropertyValueTypes

You can use the following constants with this data type.

- PropValType_Boolean
- PropValType_Container
- PropValType_NamedType
- PropValType_Number
- PropValType_Reference
- PropValType_String

Purpose

Returns the type of value and other information about the property specified by lookupString.

Return Value

Type of value stored by the property.

Parameter	Type	Description
lookupString	String	Pass an empty string to denote the PropertyObject to which the method applies, or pass the name of a subproperty within the PropertyObject. See lookupString for more details.
options	Long	Pass 0 to specify the default behavior, or pass one or more PropertyOptions constants. Use the bitwise-OR operator to specify multiple options.
isObject	Boolean	Returns True if the lookupString specifies a PropertyObject.
isArray	Boolean	Returns True if the lookupString specifies an array.
typeName	String	Returns the type name if the lookupString specifies an object that is an instance of a named type. Otherwise, returns an empty string. For a list of built-in named types see the NamedPropertyTypes constants in this help file.

See Also

PropertyOptions
NamedPropertyTypes

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

GetTypeDefinition Method

Syntax

PropertyObject.**GetTypeDefinition**(lookupString, options)

Return Type

PropertyObject

Purpose

Returns the object that is the type definition for the property specified by lookupString.

Return Value

Type definition, in the form of a PropertyObject, for the property specified by lookupString. Release it when you finish using the object.

Parameter	Type	Description
lookupString	String	Pass an empty string to denote the PropertyObject to which the method applies, or pass the name of a subproperty within the PropertyObject. See lookupString for more details.
options	Long	Pass 0 to specify the default behavior, or pass one or more PropertyOptions constants. Use the bitwise-OR operator to specify multiple options.

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

GetValBoolean Method

Syntax

PropertyObject.**GetValBoolean**(lookupString, options)

Return Type

Boolean

Purpose

Returns the Boolean value of the property specified by the lookupString parameter.

Return Value

Boolean value of the property.

Parameter	Type	Description
lookupString	String	Pass an empty string to denote the PropertyObject to which the method applies, or pass the name of a subproperty within the PropertyObject. See lookupString for more details.
options	Long	Pass 0 to specify the default behavior, or pass one or more PropertyOptions constants. Use the bitwise-OR operator to specify multiple options.

Example

```
LabWindows/CVI:
       VBOOL passFail;

       TS_PropertyGetValBoolean(propObj, &errorInfo,
                                "Step.Result.PassFail", 0, &passFail);

Visual Basic (function call):
       Dim passFail As Boolean

       passFail = propObj.GetValBoolean("Step.Result.PassFail", 0)

Visual Basic (inline):
       Dim passFail As Boolean

       passFail = propObj.Step.Result.PassFail

MFC:
       VARIANT_BOOL passFail;

       passFail = propObj.GetValBoolean("Step.Result.PassFail", 0);
```

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

GetValIDispatch Method

Syntax

PropertyObject.**GetValIDispatch**(lookupString, options)

Return Type

Object

Purpose

Returns the value of the ActiveX Automation Reference property specified by the lookupString parameter. The object is returned as an IDispatch pointer.

Remarks

Release your reference to this object when you are finished using it. This method returns a NULL reference if that is what the property contains.

Both this method and GetValInterface apply to the PropValType_Reference value type. The GetValInterface/SetValInterface methods are meant for advanced users who want to store arbitrary interfaces in TestStand reference properties.

Return Value

IDispatch pointer value of the property.

Parameter	Type	Description
lookupString	String	Pass an empty string to denote the PropertyObject to which the method applies, or pass the name of a subproperty within the PropertyObject. See lookupString for more details.
options	Long	Pass 0 to specify the default behavior, or pass one or more PropertyOptions constants. Use the bitwise-OR operator to specify multiple options.

Example

```
LabWindows/CVI:
       CAObjHandle tmpObj = 0;

       TS_PropertyGetValIDispatch(propObj, &errorInfo,
                                  "Locals.ActiveXAutoObj", 0, &tmpObj);
       ...
       CA_DiscardObjHandle(tmpObj);

Visual Basic (function call):
       Dim tmpObj As Object

       Set tmpObj = propObj.GetValIDispatch("Locals.ActiveXAutoObj", 0)

Visual Basic (inline):
       Dim tmpObj As Object

       Set tmpObj = propObj.Locals.ActiveXAutoObj

MFC:
       LPDISPATCH tmpObjPtr = NULL;

       tmpObjPtr = propObj.GetValIDispatch("Locals.ActiveXAutoObj", 0);
       ...
       tmpObjPtr->Release();
```

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

GetValInterface Method

Syntax

PropertyObject.**GetValInterface**(lookupString, options)

Return Type

LPUNKNOWN

Purpose

Returns the value of the ActiveX Automation Reference property specified by the lookupString parameter. The object is returned as an IUnknown pointer.

Remarks

Release your reference to this object when you are finished using it. This method returns a NULL reference if that is what the property contains.

Both this method and GetValIDispatch apply to the PropValType_Reference value type. The GetValInterface/SetValInterface methods are meant for advanced users who want to store arbitrary interfaces in TestStand reference properties.

Return Value

IUnknown pointer value of the property.

Parameter	Type	Description
lookupString	String	Pass an empty string to denote the PropertyObject to which the method applies, or pass the name of a subproperty within the PropertyObject. See lookupString for more details.
options	Long	Pass 0 to specify the default behavior, or pass one or more PropertyOptions constants. Use the bitwise-OR operator to specify multiple options.

Example

```
LabWindows/CVI:
       LPUNKNOWN tmpObj = NULL;

       TS_PropertyGetValInterface(propObj, &errorInfo,
                                  "Locals.ActiveXAutoObj", 0, &tmpObj);
       ...
       tmpObj->lpVtbl->Release(tmpObj);

Visual Basic (function call):
       Dim tmpObj As SpecificInterfaceType

       Set tmpObj = propObj.GetValInterface("Locals.ActiveXAutoObj", 0)

Visual Basic (inline):
       Dim tmpObj As SpecificInterfaceType

       Set tmpObj = propObj.Locals.ActiveXAutoObj

MFC:
       LPUNKNOWN tmpObjPtr = NULL;

       tmpObjPtr = propObj.GetValInterface("Locals.ActiveXAutoObj", 0);
       ...
       tmpObjPtr->Release();
```

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

GetValNumber Method

Syntax

PropertyObject.**GetValNumber**(lookupString, options)

Return Type

Double

Purpose

Returns the numeric value of the property specified by the lookupString parameter.

Remarks

The data type of the value you obtain from the method is double because TestStand stores all numeric values as 64-bit floating point values.

Return Value

Numeric value of the property.

Parameter	Type	Description
lookupString	String	Pass an empty string to denote the PropertyObject to which the method applies, or pass the name of a subproperty within the PropertyObject. See lookupString for more details.
options	Long	Pass 0 to specify the default behavior, or pass one or more PropertyOptions constants. Use the bitwise-OR operator to specify multiple options.

Example

```
LabWindows/CVI:
       double highLimit;

       TS_PropertyGetValNumber(propObj, &errorInfo, "Step.Limits.High",
                               0, &highLimit);

Visual Basic (function call):
       Dim highLimit As Double

         highLimit = propObj.GetValNumber("Step.Limits.High", 0)

Visual Basic (inline):
       Dim highLimit As Double

       highLimit = propObj.Step.Limits.High

MFC:
       double highLimit;

       highLimit = propObj.GetValNumber("Step.Limits.High", 0);
```

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

GetValString Method

Syntax

PropertyObject.**GetValString**(lookupString, options)

Return Type

String

Purpose

Returns the string value of the property specified by the lookupString parameter.

Return Value

String value of the property.

Parameter	Type	Description
lookupString	String	Pass an empty string to denote the PropertyObject to which the method applies, or pass the name of a subproperty within the PropertyObject. See lookupString for more details.
options	Long	Pass 0 to specify the default behavior, or pass one or more PropertyOptions constants. Use the bitwise-OR operator to specify multiple options.

Example

```
LabWindows/CVI:
       char *stringVal = NULL;

       TS_PropertyGetValString(propObj, &errorInfo, "Step.Limits.String",
                               0, &stringVal);
       ...
       CA_FreeMemory(stringVal);

Visual Basic (function call):
       Dim stringVal As String

       stringVal = propObj.GetValString("Step.Limits.String", 0)

Visual Basic (inline):
       Dim stringVal As String

       stringVal = propObj.Step.Limits.String

MFC:
       CString stringVal;

       stringVal = propObj.GetValString("Step.Limits.String", 0);
```

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

GetValVariant Method

Syntax

PropertyObject.**GetValVariant**(lookupString, options)

Return Type

Variant

Purpose

Returns the value of the property specified by the lookupString parameter in a variant. Use this method to get the value of an entire array at once. Refer to the example for more details.

Return Value

Variant containing the value of the property

Parameter	Type	Description
lookupString	String	Pass an empty string to denote the PropertyObject to which the method applies, or pass the name of a subproperty within the PropertyObject. See lookupString for more details.
options	Long	Pass 0 to specify the default behavior, or pass one or more PropertyOptions constants. Use the bitwise-OR operator to specify multiple options.

Example

```
LabWindows/CVI:
       VARIANT tmpVariant;
       double *numArray = NULL;
       unsigned int numElements;

       TS_PropertyGetValVariant(propObj, &errorInfo,
                                "Locals.NumArray", 0, &tmpVariant);
       CA_VariantGet1DArray(&tmpVariant, CAVT_DOUBLE, &numArray,
                            &numElements);
       ...
       CA_FreeMemory(numArray);

Visual Basic (function call):
       Dim numArray As Variant

       numArray = propObj.GetValVariant("Locals.NumArray", 0)

Visual Basic (inline):
       Dim numArray As Variant

       numArray = propObj.Locals.NumArray

MFC:
       COleSafeArray tmpSafeArray;

       tmpSafeArray.Attach(propObj.GetValVariant("Locals.NumArray", 0));
       // See your compiler's documentation for getting
       // values from a safe array.
```

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

NewSubProperty Method

Syntax

PropertyObject.**NewSubProperty**(lookupString, valueType, asArray, typeName, options)

Purpose

Creates a new subproperty with the name specified by the lookupString parameter.

Parameter	Type	Description
lookupString	String	Pass an empty string to denote the PropertyObject to which the method applies, or pass the name of a subproperty within the PropertyObject. See lookupString for more details.
valueType	PropertyValueTypes	Pass the type of value you want the new subproperty to store.
asArray	Boolean	Pass True to make the new subproperty an array whose elements are of the type you specify in valueType.
typeName	String	Pass the name of an existing type if you want to create the new subproperty as an instance of a named type. Otherwise, pass an empty string. If you pass a type name you must pass PropValType_NamedType for the valueType parameter. For a list of built-in named types see the NamedPropertyTypes constants in this help file.
options	Long	Pass 0 to specify the default behavior, or pass one or more PropertyOptions constants. Use the bitwise-OR operator to specify multiple options.

See Also

PropertyOptions
NamedPropertyTypes

Read Method

Syntax

PropertyObject.**Read**(pathString, objectName, RWoptions)

Purpose

Reads the contents of an object from a file and stores them in the PropertyObject. You specify the file with the pathString parameter and the object in the file with the objectName parameter.

Parameter	Type	Description
pathString	String	Pass the pathname of the file from which to read the object data.
objectName	String	Pass the name with which the object is stored in the file.
RWoptions	Long	Pass 0 to specify the default behavior, or pass one or more ReadWriteOptions constants. Use the bitwise-OR operator to specify multiple read/write options.

See Also

Write
ReadWriteOptions

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

Serialize Method

Syntax

PropertyObject.**Serialize**(stream, objectName, RWoptions)

Purpose

Adds the contents of the object to a stream of string data.

Parameter	Type	Description
stream	String	Pass an existing string or an empty string. The method creates a new string containing the contents of the string passed for this parameter and the contents of the object. You obtain the new string back from this parameter when the function returns.
objectName	String	Pass the name with which to store the object in the stream.
RWoptions	Long	Pass 0 to specify the default behavior, or pass one or more ReadWriteOptions constants. Use the bitwise-OR operator to specify multiple read write options.

See Also

Unserialize
ReadWriteOptions

SetDimensions Method

Syntax

PropertyObject.**SetDimensions**(lookupString, options, lowerBounds, upperBounds)

Purpose

Sets the upper and lower bounds for the array specified by the lookupString parameter.

Parameter	Type	Description
lookupString	String	Pass an empty string to denote the PropertyObject to which the method applies, or pass the name of a subproperty within the PropertyObject. See lookupString for more details.
options	Long	Pass 0 to specify the default behavior, or pass one or more PropertyOptions constants. Use the bitwise-OR operator to specify multiple options.
lowerBounds	String	An array index string describing the lowest valid index for each dimension. For example, a two-dimensional array with a lower bound of 0 for the first dimension and 2 for the second would have the lower bound string "[0][2]".
upperBounds	String	An array index string describing the highest valid index for each dimension. For example, a two-dimensional array with an upper bound of 2 for the first dimension and 4 for the second would have the upper bound string "[2][4]".

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

SetFlags Method

Syntax

PropertyObject.**SetFlags**(lookupString, options, flags)

Purpose

Sets the flags setting of the property specified by lookupString.

Parameter	Type	Description
lookupString	String	Pass an empty string to denote the PropertyObject to which the method applies, or pass the name of a subproperty within the PropertyObject. See lookupString for more details.
options	Long	Pass 0 to specify the default behavior, or pass one or more PropertyOptions constants. Use the bitwise-OR operator to specify multiple options.
flags	Long	The new flags for the property. Refer to PropertyFlags for possible values.

SetNthSubPropertyName Method

Syntax

PropertyObject.**SetNthSubPropertyName**(lookupString, index, options, newValue)

Purpose

Sets the name of a subproperty within the property specified by lookupString. You identify the property using a zero-based index.

Parameter	Type	Description
lookupString	String	Pass an empty string to denote the PropertyObject to which the method applies, or pass the name of a subproperty within the PropertyObject. See lookupString for more details.
index	Long	Pass the index of the subproperty to name. This index is zero-based.
options	Long	Pass 0 to specify the default behavior, or pass one or more PropertyOptions constants. Use the bitwise-OR operator to specify multiple options.
newValue	String	New name for the subproperty.

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

SetPropertyObject Method

Syntax

PropertyObject.**SetPropertyObject**(lookupString, options, newValue)

Purpose

Sets the subproperty specified by the lookupString parameter to the PropertyObject object that you pass.

Parameter	Type	Description
lookupString	String	Pass an empty string to denote the PropertyObject to which the method applies, or pass the name of a subproperty within the PropertyObject. See lookupString for more details.
options	Long	Pass 0 to specify the default behavior, or pass one or more PropertyOptions constants. Use the bitwise-OR operator to specify multiple options.
newValue	PropertyObject	New value for the property.

SetValBoolean Method

Syntax

PropertyObject.**SetValBoolean**(lookupString, options, newValue)

Purpose

Sets the Boolean value of the property specified by the lookupString parameter.

Parameter	Type	Description
lookupString	String	Pass an empty string to denote the PropertyObject to which the method applies, or pass the name of a subproperty within the PropertyObject. See lookupString for more details.
options	Long	Pass 0 to specify the default behavior, or pass one or more PropertyOptions constants. Use the bitwise-OR operator to specify multiple options.
newValue	Boolean	New value for the property.

Example

```
LabWindows/CVI:
      TS_PropertySetValBoolean(propObj, &errorInfo,
                               "Step.Result.PassFail", 0, VTRUE);

Visual Basic (function call):
      propObj.SetValBoolean("Step.Result.PassFail", 0, True)

Visual Basic (inline):
      propObj.Step.Result.PassFail = True

MFC:
      propObj.SetValBoolean("Step.Result.PassFail", 0, TRUE);
```

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

SetValIDispatch Method

Syntax

PropertyObject.**SetValIDispatch**(lookupString, options, newValue)

Purpose

Sets the value of the ActiveX Automation Reference property specified by the lookupString parameter. You must specify the value as an IDispatch pointer.

Remarks

The reference property keeps its own reference to the object you specify. If the property already contains a reference, it releases that reference before storing the new one. If you specify a NULL reference the property will release any existing reference it holds.

Both this method and SetValInterface apply to the PropValType_Reference value type. The GetValInterface/SetValInterface methods are meant for advanced users who want to store arbitrary interfaces in TestStand reference properties.

Parameter	Type	Description
lookupString	String	Pass an empty string to denote the PropertyObject to which the method applies, or pass the name of a subproperty within the PropertyObject. See lookupString for more details.
options	Long	Pass 0 to specify the default behavior, or pass one or more PropertyOptions constants. Use the bitwise-OR operator to specify multiple options.
newValue	Object	New value for the property.

Example

```
LabWindows/CVI:
       CAObjHandle objHandle = 0;

       objHandle = NewActiveXAutomationObject();
       TS_PropertySetValIDispatch(propObj, &errorInfo,
                                  "Locals.ActiveXAutoObj", 0, objHandle);
       CA_DiscardObjHandle(objHandle);

Visual Basic (function call):
       propObj.SetValIDispatch("Locals.ActiveXAutoObj", 0, someObj)

Visual Basic (inline):
       propObj.Locals.ActiveXAutoObj = someObj

MFC:
       propObj.SetValIDispatch("Locals.ActiveXAutoObj", 0, someObj);
```

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

SetValInterface Method

Syntax

PropertyObject.**SetValInterface**(lookupString, options, newValue)

Purpose

Sets the value of the ActiveX Automation Reference property specified by the lookupString parameter. You must specify the value as an IUnknown pointer.

Remarks

The reference property keeps its own reference to the object you specify. If the property already contains a reference, it releases that reference before storing the new one. If you specify a NULL reference the property will release any existing reference it holds.

Both this method and SetValIDispatch apply to the PropValType_Reference value type. The GetValInterface/SetValInterface methods are meant for advanced users who want to store arbitrary interfaces in TestStand reference properties.

Parameter	Type	Description
lookupString	String	Pass an empty string to denote the PropertyObject to which the method applies, or pass the name of a subproperty within the PropertyObject. See lookupString for more details.
options	Long	Pass 0 to specify the default behavior, or pass one or more PropertyOptions constants. Use the bitwise-OR operator to specify multiple options.
newValue	LPUNKNOWN	New value for the property.

Example

```
LabWindows/CVI:
       TS_PropertySetValInterface(propObj, &errorInfo,
                                  "Locals.ActiveXAutoObj", 0, someObj);

Visual Basic (function call):
       propObj.SetValInterface("Locals.ActiveXAutoObj", 0, someObj)

Visual Basic (inline):
       propObj.Locals.ActiveXAutoObj = someObj

MFC:
       propObj.SetValInterface("Locals.ActiveXAutoObj", 0, someObj);
```

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

SetValNumber Method

Syntax

PropertyObject.**SetValNumber**(lookupString, options, newValue)

Purpose

Sets the numeric value of the property specified by the lookupString parameter.

Remarks

The data type of the value you pass to the method is double because TestStand stores all numeric values as 64-bit floating point values.

Parameter	Type	Description
lookupString	String	Pass an empty string to denote the PropertyObject to which the method applies, or pass the name of a subproperty within the PropertyObject. See lookupString for more details.
options	Long	Pass 0 to specify the default behavior, or pass one or more PropertyOptions constants. Use the bitwise-OR operator to specify multiple options.
newValue	Double	New value for the property.

Example

```
LabWindows/CVI:
      TS_PropertySetValNumber(propObj, &errorInfo, "Step.Limits.High",
                              0, 1.234);

Visual Basic (function call):
      propObj.SetValNumber("Step.Limits.High", 0, 1.234)

Visual Basic (inline):
      propObj.Step.Limits.High = 1.234

MFC:
      propObj.SetValNumber("Step.Limits.High", 0, 1.234);
```

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

SetValString Method

Syntax

PropertyObject.**SetValString**(lookupString, options, newValue)

Purpose

Sets the string value of the property specified by the lookupString parameter.

Parameter	Type	Description
lookupString	String	Pass an empty string to denote the PropertyObject to which the method applies, or pass the name of a subproperty within the PropertyObject. See lookupString for more details.
options	Long	Pass 0 to specify the default behavior, or pass one or more PropertyOptions constants. Use the bitwise-OR operator to specify multiple options.
newValue	String	New value for the property.

Example

```
LabWindows/CVI:
      TS_PropertySetValString(propObj, &errorInfo,
                              "Step.Limits.String", 0, "success");

Visual Basic (function call):
      propObj.SetValString("Step.Limits.String", 0, "success")

Visual Basic (inline):
      propObj.Step.Result.PassFail = "success"

MFC:
      propObj.SetValString("Step.Limits.String", 0, "success");
```

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

SetValVariant Method

Syntax

PropertyObject.**SetValVariant**(lookupString, options, newValue)

Purpose

Sets the value of the property specified by the lookupString parameter with a variant. Use this method to set the value of an entire array at once. See the example for more details.

Parameter	Type	Description
lookupString	String	Pass an empty string to denote the PropertyObject to which the method applies, or pass the name of a subproperty within the PropertyObject. See lookupString for more details.
options	Long	Pass 0 to specify the default behavior, or pass one or more PropertyOptions constants. Use the bitwise-OR operator to specify multiple options.
newValue	Variant	New value for the property.

Example

```
LabWindows/CVI:
       VARIANT tmpVariant;
       double numArray[] = {0.123, 1.234, 12.34};

       CA_VariantSet1DArray (&tmpVariant, CAVT_DOUBLE, 3, numArray);
       TS_PropertySetValVariant(propObj, &errorInfo,
                                "Locals.NumArray", 0, tmpVariant);
       CA_VariantClear(tmpVariant);

Visual Basic (function call):
       Dim numArray(2) As Double

       numArray(0) = 0.123
       numArray(1) = 1.234
       numArray(2) = 12.34
       propObj.SetValVariant("Locals.NumArray", 0, numArray)

Visual Basic (inline):
       Dim numArray(2) As Double

       numArray(0) = 0.123
       numArray(1) = 1.234
       numArray(2) = 12.34
       propObj.Locals.NumArray = numArray

MFC:
       double numArray[] = {0.123, 1.234, 12.34};
       COleSafeArray tmpSafeArray;

       tmpSafeArray.CreateOneDim(VT_R8, 3, numArray);

       propObj.SetValVariant("Locals.NumArray", 0, tmpSafeArray);
```

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

Unserialize Method

Syntax

PropertyObject.**Unserialize**(stream, objectName, RWoptions)

Purpose

Reads the contents of an object from a stream of string data and stores the contents in the PropertyObject to which the method applies.

Parameter	Type	Description
stream	String	Pass an existing stream of data that stores the object.
objectName	String	Pass the name with which the object is stored in the stream.
RWoptions	Long	Pass 0 to specify the default behavior, or pass one or more ReadWriteOptions constants. Use the bitwise-OR operator to specify multiple read write options.

See Also

Serialize
ReadWriteOptions

Write Method

Syntax

PropertyObject.**Write**(pathString, objectName, RWoptions)

Purpose

Writes the contents of an object to the file specified by pathString and associates the name specified by objectName with the object in the file.

Parameter	Type	Description
pathString	String	Pass the pathname of the file to which to write the object data.
objectName	String	Pass the name to associate with the object.
RWoptions	Long	Pass 0 to specify the default behavior, or pass one or more ReadWriteOptions constants. Use the bitwise-OR operator to specify multiple read write options.

See Also

Read
ReadWriteOptions

PropertyOptions Constants

These constants represent the options you can use with many of the methods of the PropertyObject class. Use the bitwise-OR operator to specify more than one option for a particular method.

- **PropOption_Coerce** -- Use this option to convert the value of a propery from or to any of the supported basic types.
- **PropOption_CoerceFromBoolean** -- Use this option for implicit conversion when getting a boolean value as a string or number. You can use this option with the GetValNumber and GetValString methods.
- **PropOption_CoerceFromNumber** -- Use this option for implicit conversion when getting a numeric value as a boolean or string. You can use this option with the GetValBoolean and GetValString methods.

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

- **PropOption_CoerceFromReference** -- Use this option for implicit conversion when getting an ActiveX reference value as a string or number. You can use this option with the GetValNumber and GetValString methods.
- **PropOption_CoerceFromString** -- Use this option for implicit conversion when getting a string value as a boolean or number. You can use this option with the GetValNumber and GetValBoolean methods.
- **PropOption_CoerceToBoolean** -- Use this option for implicit conversion when setting a boolean value with a string or number. You can use this option with the SetValNumber and SetValString methods.
- **PropOption_CoerceToNumber** -- Use this option for implicit conversion when setting a numeric value with a boolean or string. You can use this option with the SetValBoolean and SetValString methods.
- **PropOption_CoerceToReference** -- Use this option for implicit conversion when setting an ActiveX reference value with a string or number. You can use this option with the SetValNumber and SetValString methods.
- **PropOption_CoerceToString** -- Use this option for implicit conversion when setting a string value with a boolean or number. You can use this option with the SetValNumber and SetValBoolean methods.
- **PropOption_DeleteIfExists** -- When calling PropertyObject.DeleteSubProperty, use this option to avoid getting an error if the subproperty you specify does not exist.
- **PropOption_DoNothingIfExists** -- Use this option with the SetVal methods of the PropertyObject class to avoid setting the value of a property that already exists. Usually, you do this in combination with the PropOption_InsertIfMissing option.
- **PropOption_InsertIfMissing** -- Use this option with the SetVal methods of the PropertyObject class to create a new property if the property specified by the lookupString does not exist.
- **PropOption_SetOnlyIfDoesNotExist** -- Use this constant as a shortcut for specifying both the PropOption_DoNothingIfExists and the PropOption_InsertIfMissing options.

See Also

PropertyObject

PropertyValueTypes Enumeration

These constants indicate the type of value that a PropertyObject stores.

You can use the following constants with this data type.

- PropValType_Boolean
- PropValType_Container
- PropValType_NamedType
- PropValType_Number
- PropValType_Reference
- PropValType_String

See Also

PropertyObject.GetType
PropertyObject.NewSubProperty
Engine.NewPropertyObject

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

ReadWriteOptions Constants

Use these constants to specify optional behavior for reading and writing PropertyObject objects.

- **RWOption_EraseAll** -- When writing an object to a file, use this option to clear the entire contents of the existing file.
- **RWOption_EraseExistingObject** -- When writing an object, use this option to clear out any existing object with the same name.
- **RWOption_ValuesOnly** -- Use this option to write only the value of the object and the value of each of its subproperties. TestStand does not write the type information for the object and its subproperties when you use this flag.

See Also

PropertyObject.Read
PropertyObject.Write
PropertyObject.Serialize
PropertyObject.Unserialize

Report

Use objects of the Report class to modify, save, load, retrieve, and view reports. To get the Report object for an execution, retrieve the value of the Execution.Report property.

Usually, the process model generates a report object, and the sequence editor or run-time operator interface displays it.

Properties

All	Format	Id
Location		

Methods

Append	AsPropertyObject	GetTempFile
LaunchViewer	Load	Reset
Save		

See Also

Execution.Report

All Property (Read Only)

Syntax

Report.**All**()

Data Type

String

Purpose

Returns the entire report as a string.

Remarks

If you are using the TestStand API from LabWindows/CVI, you must free the string using the CA_FreeMemory() function of the LabWindows/CVI ActiveX library.

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

Format Property

Syntax

Report.**Format**()

Data Type

String

Purpose

Use this property to get or set the file format of the report. You specify the file format by passing the file extension that commonly stands for the format. For example, txt for ASCII text files, htm or html for HTML files, and rtf for rich text format files.

Id Property (Read Only)

Syntax

Report.**Id**()

Data Type

Long

Purpose

A unique ID that distinguishes this report from all other reports.

Location Property

Syntax

Report.**Location**()

Data Type

String

Purpose

Use this property to get or set a string that specifies the report location.

Remarks

This property is automatically set to the pathname of the report file if you call the Load or Save method of this class.

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

Append Method

Syntax

Report.**Append**(stringToAppend)

Return Type

Long

Purpose

Appends a string to the report.

Return Value

Returns the number of times you appended a string to the report since you last reset it.

Parameter	Type	Description
stringToAppend	String	The string to append to the report.

AsPropertyObject Method

Syntax

Report.**AsPropertyObject**()

Return Type

PropertyObject

Purpose

Returns the underlying PropertyObject that represents the Report object. Use the PropertyObject to modify, add, or remove your own custom properties of the object.

Remarks

Do not use this function to remove or modify the dynamic properties that TestStand already provides for Report objects. You can do this through the other methods of the EditArgs class.

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

GetTempFile Method

Syntax

Report.**GetTempFile**(linefeedConversion [, extensionString])

Return Type

String

Purpose

Stores the report in a temporary file and returns the pathname of that file.

Return Value

The pathname of the temporary file this method uses to store the report.

Parameter	Type	Description
linefeedConversion	ReportConversion	Specifies a linefeed conversion for the report text.
extensionString	Variant	An optional string parameter to specify the file extension for the temporary file. If you do not specify an extension string, the Format property of the report determines the file extension.

LaunchViewer Method

Syntax

Report.**LaunchViewer**(linefeedConversion)

Purpose

Launches an external file viewer to allow the user to view the report.

Remarks

If the current TestStand configuration specifies an external viewer for the format, TestStand launches that viewer. Otherwise, TestStand launches the viewer that Windows associates with the file extension that the report format specifies.

Parameter	Type	Description
linefeedConversion	ReportConversion	Specifies a linefeed conversion for the report text.

Load Method

Syntax

Report.**Load**(pathString, linefeedConversion)

Purpose

Replaces the current report data with the data in the file you specify.

Parameter	Type	Description
pathString	String	Specifies the pathname of the file that contains the report data.
linefeedConversion	ReportConversion	Specifies a linefeed conversion for the report text.

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

Reset Method

Syntax

Report.**Reset**(newValue)

Return Type

Long

Purpose

Replaces the current report data with the data in the string you specify.

Return Value

Returns the number of times you have reset the report.

Parameter	Type	Description
newValue	String	The new report string.

Save Method

Syntax

Report.**Save**(pathString, appendIfAlreadyExists, linefeedConversion)

Purpose

Saves the report to the file you specify.

Parameter	Type	Description
pathString	String	Specifies the file to save the report in.
appendIfAlreadyExists	Boolean	Indicates whether to append or overwrite the file if it already exists. Pass True to append to the file. Pass False to overwrite it.
linefeedConversion	ReportConversion	Specifies a linefeed conversion for the report text.

ReportConversion Enumeration

This data type contains values that specify how to handle linefeeds and carriage returns in report text.

You can use the following constants with this data type.

- **ReportConv_FromCRLF** -- Convert carriage return/linefeed combinations to linefeeds. You usually use this option when reading a report from disk into memory.
- **ReportConv_NoConversion** -- Do not convert carriage returns and linefeeds.
- **ReportConv_ToCRLF** -- Convert each linefeed to a carriage return followed by a linefeed. You usually use this option when writing a report from memory to disk.

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

RTEOptions Enumeration

This data type contains values that specify how to respond to a run-time error.

You can use the following constants with this data type.

- RTEOption_Abort
- RTEOption_Continue
- RTEOption_Ignore

RunModes Constants

These constants represent the run modes you can set on a step.

- RunMode_ForceFail
- RunMode_ForcePass
- RunMode_Normal
- RunMode_Skip

See Also

Step.RunMode
Step.RunTimeRunMode

SeqFileCallbacks Constants

These constants are the names of the sequence file callback sequences you can override using SequenceFile.CreateCallbackOverrideSequence.

- SeqFileCback_Load
- SeqFileCback_PostInteractive
- SeqFileCback_PostStep
- SeqFileCback_PreInteractive
- SeqFileCback_PreStep
- SeqFileCback_Unload

See Also

SequenceFile.CreateCallbackOverrideSequence

Sequence

Objects of the Sequence class represent a sequence that can contain steps. Sequences have three groups of steps. The StepGroups enumeration contains a value for each of the step groups. SequenceTypes constants define several types of sequences. You can get a reference to the sequence objects a sequence file contains by calling the SequenceFile.GetSequence method. You can create new sequences by calling the Engine.NewSequence or SequenceFile.CreateCallbackOverrideSequence methods. Use the Sequence class to examine or modify sequence settings and to examine or modify the list of steps in the sequence.

Properties

DisableResults	EntryPointInitiallyHidden	GotoCleanupOnFailure
Locals	Name	Parameters
ShowEntryPointForAllWindows	ShowEntryPointForEditorOnly	ShowEntryPointForExeWindow
ShowEntryPointForFileWindow	Type	

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

Methods

AsPropertyObject	DeleteStep	EvalEntryPointEnabledExpression
EvalEntryPointNameExpression	GetEntryPointMenuFromHint	GetNumSteps
GetStep	GetStepByName	GetStepIndex
InsertStep	RemoveStep	StepNameExists

See Also

- Step
- StepGroups
- SequenceTypes
- SequenceFile.GetSequence
- Engine.NewSequence
- SequenceFile.CreateCallbackOverrideSequence

DisableResults Property

Syntax

Sequence.**DisableResults**()

Data Type

Boolean

Purpose

If set to True, TestStand does not record results for any steps in the sequence. If set to False, TestStand records results based on the setting of the RecordResult property of each individual step or the DisableResults property of the engine.

EntryPointInitiallyHidden Property

Syntax

Sequence.**EntryPointInitiallyHidden**()

Data Type

Boolean

Purpose

This property applies only to an entry point sequence in a process model file. The property indicates whether an execution of the entry point is hidden when it starts. In other words, if the property is True, the operator interface does not display an execution window for the entry point sequence when it begins executing it. The operator interface does this by assigning the ExecTypeMask_InitiallyHidden value to the TypeMask property of the execution.

See Also

- Execution.TypeMask
- ExecutionTypeMask

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

GotoCleanupOnFailure Property

Syntax

Sequence.**GotoCleanupOnFailure**()

Data Type

Boolean

Purpose

If this property is True, the flow of execution in the sequence jumps to the Cleanup step group whenever a step sets the status property of the sequence to Failed.

Remarks

If the Engine.AlwaysGotoCleanupOnFailure property is True, the flow of execution jumps to the Cleanup step group on failure regardless of the state of this property.

Locals Property (Read Only)

Syntax

Sequence.**Locals**()

Data Type

PropertyObject

Purpose

Returns the PropertyObject that contains the local variables for the sequence. Release your reference to this object when you are done using it.

Name Property

Syntax

Sequence.**Name**()

Data Type

String

Purpose

Use this property to get or set the name of the sequence.

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

Parameters Property (Read Only)

Syntax

Sequence.**Parameters**()

Data Type

PropertyObject

Purpose

Returns the PropertyObject that contains the parameters for the sequence. Release your reference to this object when you are done using it.

ShowEntryPointForAllWindows Property

Syntax

Sequence.**ShowEntryPointForAllWindows**()

Data Type

Boolean

Purpose

This property applies only to an entry point sequence in a process model file. Set this property to True if you want the entry point to appear in the menu bar no matter what type of window is active.

ShowEntryPointForEditorOnly Property

Syntax

Sequence.**ShowEntryPointForEditorOnly**()

Data Type

Boolean

Purpose

This property applies only to an entry point sequence in a process model file. Set this property to True if you want the entry point to appear in the menu bar of the sequence editor only and not in an operator interface.

ShowEntryPointForExeWindow Property

Syntax

Sequence.**ShowEntryPointForExeWindow**()

Data Type

Boolean

Purpose

This property applies only to an entry point sequence in a process model file. Set this property to True if you want the entry point to appear in the menu bar when an execution window is the active window.

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

ShowEntryPointForFileWindow Property

Syntax

Sequence.**ShowEntryPointForFileWindow**()

Data Type

Boolean

Purpose

This property applies only to an entry point sequence in a process model file. Set this property to True if you want the entry point to appear in the menu bar when a sequence file window is the active window.

Type Property

Syntax

Sequence.**Type**()

Data Type

SequenceTypes

You can use the following constants with this data type.

- SeqType_Callback
- SeqType_CfgEntryPoint
- SeqType_ExeEntryPoint
- SeqType_Normal

Purpose

Use this property to get or set the type of the sequence.

AsPropertyObject Method

Syntax

Sequence.**AsPropertyObject**()

Return Type

PropertyObject

Purpose

Returns the underlying PropertyObject that represents the Sequence object. Use the PropertyObject to modify, add, or remove custom properties of the object.

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

DeleteStep Method

Syntax

Sequence.**DeleteStep**(index, stepGroupParam)

Purpose

Deletes a step from the sequence.

Parameter	Type	Description
index	Long	The zero-based step index that indicates the position of the step in the step group
stepGroupParam	StepGroups	Specifies the step group that contains the step you want to delete.

EvalEntryPointEnabledExpression Method

Syntax

Sequence.**EvalEntryPointEnabledExpression**(sequenceFileParam)

Return Type

Boolean

Purpose

This method applies only to an entry point sequence in a process model file. Call this function to determine whether to enable the entry point in the operator interface menu bar. The function evaluates the Entry Point Enabled expression for the entry point sequence in the context of the sequence file that is currently selected in the operator interface.

Return Value

Returns True if the entry point should be enabled in the operator interface menu bar.

Parameter	Type	Description
sequenceFileParam	SequenceFile	The sequence file that is currently selected in the operator interface.

EvalEntryPointNameExpression Method

Syntax

Sequence.**EvalEntryPointNameExpression**(sequenceFileParam)

Return Type

String

Purpose

This method applies only to an entry point sequence in a process model file. Call this function to determine the name to display for the entry point in the operator interface menu bar. The function evaluates the Entry Point Name expression for the entry point sequence in the context of the sequence file that is currently selected in the operator interface.

Return Value

Returns the name of the entry point.

Parameter	Type	Description
sequenceFileParam	SequenceFile	The sequence file that is currently selected in the operator interface.

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

GetEntryPointMenuFromHint Method

Syntax

Sequence.**GetEntryPointMenuFromHint**(menuNameList)

Return Type

Long

Purpose

This method applies only to an entry point sequence in a process model file. Call this function to determine the operator interface menu in which the entry point belongs. The method uses the Menu Hints option for the entry point sequence.

Return Value

Returns a zero-based index into the list of menu names that you pass as the menuNameList parameter. This index indicates the menu that the entry point belongs in. If none of the menus in your list matches a menu hint for the entry point, the method returns -1. If that occurs, you must choose the menu in which to display the entry point. Usually, you can do this based on the type of entry point.

Parameter	Type	Description
menuNameList	String	Pass a comma-separated list of the menu names in your operator interface. For example: "File,Configure,Debug". The method ignores all '_' and '&' characters when comparing the names in the list with the menu hints for the entry point.

GetNumSteps Method

Syntax

Sequence.**GetNumSteps**(stepGroupParam)

Return Type

Long

Purpose

Returns the number of steps in the specified step group of the sequence.

Parameter	Type	Description
stepGroupParam	StepGroups	Specifies a particular step group.

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

GetStep Method

Syntax

Sequence.**GetStep**(index, stepGroupParam)

Return Type

Step

Purpose

Returns a reference to the Step object that you specify by an index. Release your reference to this object when you are done using it.

Parameter	Type	Description
index	Long	The zero-based step index that indicates the position of the step in the step group.
stepGroupParam	StepGroups	Specifies a particular step group.

See Also

Step

GetStepByName Method

Syntax

Sequence.**GetStepByName**(stepName, stepGroupParam)

Return Type

Step

Purpose

Returns a reference to the Step object you specify by name. Release your reference to this object when you are done using it.

Remarks

Returns an error if no Step exists with the name you specify.

Parameter	Type	Description
stepName	String	The name of the step you want a reference to. If there is more than one step with the same name in the step group, the method returns the first step that has the name.
stepGroupParam	StepGroups	Specifies a particular step group.

See Also

Step
StepNameExists
GetStepIndex

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

GetStepIndex Method

Syntax

Sequence.**GetStepIndex**(stepName, stepGroupParam)

Return Type

Long

Purpose

Returns the index of the step with the name that matches the name you specify.

Return Value

Returns the index of the step with the name that matches the name you specify. Returns -1 if no such step exists.

Parameter	Type	Description
stepName	String	The name of the step you want the index of. If there is more than one step with the same name in the step group, the method returns the index of the first step that has the name.
stepGroupParam	StepGroups	Specifies a particular step group.

See Also

StepNameExists
GetStepByName

InsertStep Method

Syntax

Sequence.**InsertStep**(stepToInsert, index, stepGroupParam)

Purpose

Inserts a step into the sequence.

Remarks

Never insert a step into a sequence when the step resides in another sequence. The step reference you pass must be the only reference to the step. You can obtain the sole reference to a step by calling Engine.NewStep or Sequence.RemoveStep.

Parameter	Type	Description
stepToInsert	Step	The step to insert.
index	Long	The zero-based index that specifies the location in the step group at which to insert the step.
stepGroupParam	StepGroups	Specifies a particular step group.

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

RemoveStep Method

Syntax

Sequence.**RemoveStep**(index, stepGroupParam)

Return Type

Step

Purpose

Removes a step from a sequence and returns a reference to it. Release your reference to the step when you are done using it.

Parameter	Type	Description
index	Long	The zero-based index that specifies the location in the step group at which to insert the step.
stepGroupParam	StepGroups	Specifies a particular step group.

StepNameExists Method

Syntax

Sequence.**StepNameExists**(stepName, stepGroupParam)

Return Type

Boolean

Purpose

Returns True if a step with the name you specify exists in the step group.

Parameter	Type	Description
stepName	String	The name to search for.
stepGroupParam	StepGroups	Specifies a particular step group.

See Also

GetStepByName
GetStepIndex

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

SequenceCallStepAdditions Constants

Use these string constants to create lookupStrings when using the PropertyObject class to access the adapter-specific properties of a Sequence Call step.

- **SCStep_ActualArgsProp** -- An object that contains a list of arguments to the subsequence. If the subsequence has fewer parameters than the number of arguments, the subsequence ignores the extra arguments. If the subsequence has more parameters than the number of arguments, the subsequences uses the default values of the extra parameters.
- **SCStep_ArgPrototypeProp** -- An object that contains a list of parameters. TestStand uses the parameter list only for displaying the arguments in the SpecifyModule dialog box. The SpecifyModule dialog box sets this property when you select a sequence with the "Load Prototype" button. The value of this property is meaningful only when SCStep_UseArgPrototypeProp is True.
- **SCStep_IgnoreTerminateProp** -- If this Boolean property is True and the user or a step module attempts to terminate the subsequence before it completes, the execution continues with the next step.
- **SCStep_SeqFilePathExprProp** -- An expression to evaluate to obtain the pathname of the sequence file that contains the subsequence. Must evaluate to a string. The value of this property is meaningful only when SCStep_SpecifyByExprProp is True and Step_UseCurrentFileProp is False.
- **SCStep_SeqFilePathProp** -- Pathname of the sequence file that contains the subsequence. The value of this property is meaningful only when SCStep_SpecifyByExprProp is False and SCStep_UseCurrentFileProp is False.
- **SCStep_SeqNameExprProp** -- Expression to evaluate to obtain the name of the subsequence to be called. Must evaluate to a string. The value of this property is meaningful only when SCStep_SpecifyByExprProp is True.
- **SCStep_SeqNameProp** -- Name of the subsequence. The value of this property is meaningful only when SCStep_SpecifyByExprProp is False.
- **SCStep_SpecifyByExprProp** -- If this Boolean property is True, TestStand evaluates the SCStep_SeqFilePathExprProp expression property to determine the pathname of the sequence file that contains the subsequence; TestStand also evaluates the SCStep_SeqNameExprProp expression property to determine the name of the subsequence.
- **SCStep_TraceSettingProp** -- A property that controls whether tracing occurs while the subsequence executes. The three possible values are: SCStep_TraceSettingValOff, Step_TraceSettingValOn, and SCStep_TraceSettingValDontChange.
- **SCStep_TraceSettingValDontChange** -- A value of the SCStep_TraceSettingProp property. It causes tracing to occur while the subsequence executes only when tracing is enabled in the Sequence Call step that calls the subsequence.
- **SCStep_TraceSettingValOff** -- A value of the SCStep_TraceSettingProp property. It disables tracing while the subsequence executes.
- **SCStep_TraceSettingValOn** -- A value of the SCStep_TraceSettingProp property. It enables tracing while the subsequence executes.
- **SCStep_UseArgPrototypeProp** -- If this Boolean property is True, the prototype for displaying arguments in the SpecifyModule dialog box is taken from the SCStep_ArgPrototypeProp property. If this property is false, the prototype is taken from the selected sequence, if any. The SpecifyModule dialog sets this property to true when you check the "Specify Expressions for Pathname and Sequence" checkbox or when you uncheck the "Use Prototype of Selected Sequence" checkbox.
- **SCStep_UseCurrentFileProp** -- A Boolean property that is True when the file that contains the Sequence Call step also contains the subsequence.

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

SequenceContext

A SequenceContext object contains complete information about an execution at a particular point during the execution. If you use the LabVIEW Standard Prototype Adapter or the C/CVI Standard Prototype Adapter for your step module, TestStand gives you the option of passing a sequence context to your LabVIEW VI or C function. If you use the DLL Flexible Prototype Adapter, you can pass the sequence context or its subproperties as parameters. You can use the sequence context to access all the objects, variables, and properties in the execution. From the sequence context, you also can obtain references to all the steps in the current sequence, the sequence contexts for the calling sequences, the process model entry point sequence, and the main sequence in the client sequence file.

Properties

ApplicationIsEditor	Caller	CallStackDepth
CallStackName	Engine	Execution
FileGlobals	InInteractiveMode	InteractiveContext
IsProcessModel	Locals	LoopIndex
LoopNumFailed	LoopNumPassed	Main
NextStep	NextStepIndex	Parameters
PreviousStep	PreviousStepIndex	ProcessModelClient
Report	Root	RunTimeErrorMessage
SelectedExecution	SelectedFile	SelectedSequences
SelectedSteps	Sequence	SequenceFailed
SequenceFile	SequenceIndex	StationGlobals
Step	StepGroup	StepGroupStartedInteractiveExe
StepIndex	Thread	Tracing

Methods

AsPropertyObject	IsInteractiveStep

ApplicationIsEditor Property (Read Only)

Syntax

SequenceContext.**ApplicationIsEditor**()

Data Type

Boolean

Purpose

Returns True if the current application is a sequence editor.

Caller Property (Read Only)

Syntax

SequenceContext.**Caller**()

Data Type

SequenceContext

Purpose

Returns the sequence context of the calling sequence.

Remarks

Returns a NULL reference if no calling sequence exists.

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

CallStackDepth Property (Read Only)

Syntax

SequenceContext.**CallStackDepth**()

Data Type

Long

Purpose

Returns the sequence call depth of this sequence context in the current execution.

CallStackName Property (Read Only)

Syntax

SequenceContext.**CallStackName**()

Data Type

String

Purpose

Returns the name to display in the call stack list for the sequence context.

Engine Property (Read Only)

Syntax

SequenceContext.**Engine**()

Data Type

Engine

Purpose

Returns a reference to the Engine object. Release your reference to this object when you are done using it.

Execution Property (Read Only)

Syntax

SequenceContext.**Execution**()

Data Type

Execution

Purpose

Returns a reference to the currently executing execution. Release your reference to this object when you are done using it.

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

FileGlobals Property (Read Only)

Syntax

SequenceContext.**FileGlobals**()

Data Type

PropertyObject

Purpose

Returns a reference to the object that contains the globals for the currently executing sequence file.

Remarks

Release your reference to this object when you are done using it.

InInteractiveMode Property (Read Only)

Syntax

SequenceContext.**InInteractiveMode**()

Data Type

Boolean

Purpose

Returns True if the sequence context is for an interactive execution.

InteractiveContext Property (Read Only)

Syntax

SequenceContext.**InteractiveContext**()

Data Type

InteractiveContext

Purpose

If the sequence context is for an interactive execution, this property returns a reference to an InteractiveContext object. Release your reference to this object when you are done using it.

Remarks

Returns a NULL reference if the sequence context is not for an interactive execution.

IsProcessModel Property (Read Only)

Syntax

SequenceContext.**IsProcessModel**()

Data Type

Boolean

Purpose

Returns True if the current execution is a process model entry point execution.

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

Locals Property (Read Only)

Syntax

SequenceContext.**Locals**()

Data Type

PropertyObject

Purpose

Returns a reference to the object that contains the locals for the currently executing sequence.

Remarks

Release your reference to this object when you are done using it.

LoopIndex Property

Syntax

SequenceContext.**LoopIndex**()

Data Type

Long

Purpose

Use this property to store the number of iterations completed so far when implementing loop expressions for a step.

LoopNumFailed Property

Syntax

SequenceContext.**LoopNumFailed**()

Data Type

Long

Purpose

This property stores the number of failed instances when implementing loop expressions for a step.

LoopNumPassed Property

Syntax

SequenceContext.**LoopNumPassed**()

Data Type

Long

Purpose

This property stores the number of passed instances when implementing loop expressions for a step.

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

Main Property (Read Only)

Syntax

SequenceContext.**Main**()

Data Type

SequenceContext

Purpose

Returns the main sequence context of the execution with which this sequence context is associated.

Remarks

If you start an execution using a process model entry point, the main sequence context is the sequence context of the MainSequence in the client sequence file. If you start an execution without using an execution entry point, by executing a sequence or steps directly, then the main sequence context is the same as the root sequence context stored in the SequenceContext.Root property. Release your reference to this object when you are done using it.

See Also

Root

NextStep Property (Read Only)

Syntax

SequenceContext.**NextStep**()

Data Type

Step

Purpose

Returns a reference to the next step to execute. Release your reference to this object when you are done using it.

Remarks

Returns a NULL reference if there is no next step.

NextStepIndex Property

Syntax

SequenceContext.**NextStepIndex**()

Data Type

Long

Purpose

Use this property to set or get the zero-based index of the next step to execute. The index indicates the position of the step in the step group that the StepGroup property identifies.

Remarks

This property returns -1 if the step group contains no additional steps to execute.

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

Parameters Property (Read Only)

Syntax

SequenceContext.**Parameters**()

Data Type

PropertyObject

Purpose

Returns a reference to the object that contains the parameters for the currently executing sequence.

Remarks

Release your reference to this object when you are done using it.

PreviousStep Property (Read Only)

Syntax

SequenceContext.**PreviousStep**()

Data Type

Step

Purpose

Returns a reference to the Step that executed last. Release your reference to this object when you are done using it.

Remarks

Returns a NULL reference if there is no previous step.

PreviousStepIndex Property (Read Only)

Syntax

SequenceContext.**PreviousStepIndex**()

Data Type

Long

Purpose

Returns the zero-based index of the step that executed last. The index indicates the position of the step in the step group that the StepGroup property identifies.

Remarks

This property returns -1 if no step executed previously.

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

ProcessModelClient Property (Read Only)

Syntax

SequenceContext.**ProcessModelClient**()

Data Type

SequenceFile

Purpose

Returns a reference to the client sequence file of an process model execution. Release your reference to this object when you are done using it.

Remarks

Before calling this function, check the SequenceContext.IsProcessModel property to determine whether this sequence context is for a process model execution.

See Also

IsProcessModel

Report Property (Read Only)

Syntax

SequenceContext.**Report**()

Data Type

Report

Purpose

Returns a reference to the current Report. Release your reference to this object when you are done using it.

Root Property (Read Only)

Syntax

SequenceContext.**Root**()

Data Type

SequenceContext

Purpose

Returns the root sequence context of the execution with which this sequence context is associated.

Remarks

The root sequence context is the first sequence context of the execution. Release your reference to this object when you are done using it.

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

RunTimeErrorMessage Property (Read Only)

Syntax

SequenceContext.**RunTimeErrorMessage**()

Data Type

String

Purpose

If a run-time error occurs, use this property to get the error message describing the run-time error.

SelectedExecution Property (Read Only)

Syntax

SequenceContext.**SelectedExecution**()

Data Type

Execution

Purpose

Returns a reference to the execution, if any, that was selected when this execution began.

Remarks

Release your reference to this object when you are done using it.

SelectedFile Property (Read Only)

Syntax

SequenceContext.**SelectedFile**()

Data Type

SequenceFile

Purpose

Returns a reference to the sequence file, if any, that was selected when this execution began.

Remarks

Release your reference to this object when you are done using it.

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

SelectedSequences Property (Read Only)

Syntax

SequenceContext.**SelectedSequences**()

Data Type

Variant

Purpose

Returns an array of references to the sequences, if any, that were selected when this execution began.

Remarks

Returns an empty array if no sequences were selected.

See Also

Sequence

SelectedSteps Property (Read Only)

Syntax

SequenceContext.**SelectedSteps**()

Data Type

Variant

Purpose

Returns an array of references to the steps, if any, that were selected when this execution began.

Remarks

Returns an empty array if no steps were selected.

See Also

Step

Sequence Property (Read Only)

Syntax

SequenceContext.**Sequence**()

Data Type

Sequence

Purpose

Returns a reference to the run-time copy of the sequence that is currently executing. Release your reference to this object when you are done using it.

See Also

Sequence

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

SequenceFailed Property

Syntax

SequenceContext.**SequenceFailed**()

Data Type

Boolean

Purpose

Use this property to get or set the failure state of the currently executing sequence.

SequenceFile Property (Read Only)

Syntax

SequenceContext.**SequenceFile**()

Data Type

SequenceFile

Purpose

Returns a reference to the sequence file of the sequence that is executing. Release your reference to this object when you are done using it.

See Also

SequenceFile

SequenceIndex Property (Read Only)

Syntax

SequenceContext.**SequenceIndex**()

Data Type

Long

Purpose

Returns the zero-based index of the currently executing sequence in the sequence file.

StationGlobals Property (Read Only)

Syntax

SequenceContext.**StationGlobals**()

Data Type

PropertyObject

Purpose

Returns a reference to the object that contains the globals for the current installation of TestStand.

Remarks

Release your reference to this object when you are done using it.

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

Step Property (Read Only)

Syntax

SequenceContext.**Step**()

Data Type

Step

Purpose

Returns a reference to the currently executing Step. Release your reference to this object when you are done using it.

Remarks

Returns NULL if the execution is currently suspended at a breakpoint.

See Also

Step

StepGroup Property (Read Only)

Syntax

SequenceContext.**StepGroup**()

Data Type

StepGroups

You can use the following constants with this data type.

- StepGroup_Cleanup
- StepGroup_Main
- StepGroup_Setup

Purpose

Returns the currently executing step group.

StepGroupStartedInteractiveExe Property (Read Only)

Syntax

SequenceContext.**StepGroupStartedInteractiveExe**()

Data Type

Boolean

Purpose

Returns True if the current step group is the one in which an interactive execution was started and this is the sequence context for that execution.

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

StepIndex Property (Read Only)

Syntax
SequenceContext.**StepIndex**()

Data Type
Long

Purpose
Returns the zero-based index of the currently executing step. The index indicates the position of the step in the step group that the StepGroup property identifies. If execution is suspended between steps, this property returns -1.

Thread Property (Read Only)

Syntax
SequenceContext.**Thread**()

Data Type
Thread

Purpose
Returns a reference to the currently executing thread. Release your reference to this object when you are done using it.

Tracing Property

Syntax
SequenceContext.**Tracing**()

Data Type
Boolean

Purpose
Use this property to set or get a Boolean value indicating whether to trace the execution of steps to which this sequence context applies.

AsPropertyObject Method

Syntax
SequenceContext.**AsPropertyObject**()

Return Type
PropertyObject

Purpose
Returns the underlying PropertyObject that represents the SequenceContext object. Use the PropertyObject to modify, add, or remove custom properties of the object.

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

IsInteractiveStep Method

Syntax

SequenceContext.**IsInteractiveStep**(stepIndexParam)

Return Type

Boolean

Purpose

Returns True if the execution that the sequence context represents is in interactive mode and the step you specify is selected for interactive execution.

Parameter	Type	Description
stepIndexParam	Long	Pass the zero-based index of the step within the current step group.

SequenceContextProperties Constants

Use these constants to create lookupStrings to access some of the commonly used, built-in properties of a sequence context using the PropertyObject class.

- RunState_LoopIndex
- RunState_LoopNumFailed
- RunState_LoopNumPassed
- RunState_SeqFileProp
- RunState_SeqProp
- RunState_StepProp
- SeqContext_RunStateProp

SequenceFile

Objects of the SequenceFile class represent a sequence file that can contain sequences. You can obtain a reference to a sequence file object by using the Engine.GetSequenceFile or Engine.NewSequenceFile methods. Use the reference to examine or modify sequence file settings and to examine or modify the list of sequences in the sequence file.

Properties

ChangeCount	FileGlobalsDefaultValues	HasModel
IsExecuting	ModuleLoadOption	ModuleUnloadOption
NumSequences	Path	UnloadCallbackEnabled

Methods

AsPropertyObject	CreateCallbackOverrideSequence	DeleteSequence
GetModelSequenceFile	GetSequence	GetSequenceByName
GetSequenceIndex	IncChangeCount	InsertSequence
NewEditContext	RemoveSequence	Save
SequenceNameExists		

See Also

Sequence
Engine.GetSequenceFile
Engine.NewSequenceFile

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

ChangeCount Property (Read Only)

Syntax

SequenceFile.**ChangeCount**()

Data Type

Long

Purpose

Returns the number of times modifications have been made to the sequence file.

Remarks

The sequence editor and operator intefaces use this count to determine when to refresh sequence displays and when to indicate to the user that a sequence file has been modified.

See Also

IncChangeCount

FileGlobalsDefaultValues Property (Read Only)

Syntax

SequenceFile.**FileGlobalsDefaultValues**()

Data Type

PropertyObject

Purpose

Returns the PropertyObject that contains the global variables for the sequence file. Release your reference to this object when you are finished using it.

HasModel Property (Read Only)

Syntax

SequenceFile.**HasModel**()

Data Type

Boolean

Purpose

Returns True if TestStand associates a process model sequence file with the sequence file on which you call the method.

See Also

GetModelSequenceFile

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

IsExecuting Property (Read Only)

Syntax

SequenceFile.**IsExecuting**()

Data Type

Boolean

Purpose

Returns True if the sequence file contains one or more sequences that are currently active in an execution.

Remarks

If this property returns True, do not allow the user to edit the sequence file in the operator interface.

ModuleLoadOption Property

Syntax

SequenceFile.**ModuleLoadOption**()

Data Type

ModuleLoadOptions

You can use the following constants with this data type.

- **LoadOption_DynamicLoad** -- Do not load the code module for a step until the step is ready to call it.
- **LoadOption_PreloadWhenExecuted** -- Load the code module for a step when any sequence in the sequence file containing the step begins executing.
- **LoadOption_PreloadWhenOpened** -- Load the code module for a step when TestStand loads into memory the sequence file containing the step.
- **LoadOption_UseStepLoadOption** -- Load each code module according to the load option for the step that uses it. This option is valid only for the SequenceFile.ModuleLoadOption property.

Purpose

Use this property to get or set the option that instructs TestStand when to load the code modules called by the steps in the sequence file. You can use this property to override the load option of all steps in the sequence file, or you can defer to the load option contained in each individual step.

See Also

ModuleUnloadOption
Step.ModuleLoadOption

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

ModuleUnloadOption Property

Syntax

SequenceFile.**ModuleUnloadOption**()

Data Type

ModuleUnloadOptions

You can use the following constants with this data type.

- **UnloadOption_AfterSequenceExecution** -- Unload the code module for a step after the sequence containing the step finishes executing.
- **UnloadOption_AfterStepExecution** -- Unload the code module for a step after the step finishes executing.
- **UnloadOption_OnPreconditionFailure** -- Unload the code module for a step if the precondition for the step evaluates to False.
- **UnloadOption_UseStepUnloadOption** -- Unload the code module for a step according to the unload option for the step. This option is valid only for the SequenceFile.ModuleUnloadOption property.
- **UnloadOption_WithSequenceFile** -- Unload the code module for a step when TestStand unloads from memory the sequence file containing the step.

Purpose

Use this property to get or set the option that instructs TestStand when to unload the code modules called by the steps in the sequence file. You can use this property to override the unload option of all steps in the sequence file, or you can defer to the unload option contained in each individual step.

See Also

ModuleLoadOption
Step.ModuleUnloadOption

NumSequences Property (Read Only)

Syntax

SequenceFile.**NumSequences**()

Data Type

Long

Purpose

Returns the number of sequences contained in the sequence file.

Path Property (Read Only)

Syntax

SequenceFile.**Path**()

Data Type

String

Purpose

Returns the pathname TestStand used when the sequence file was last saved or loaded.

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

UnloadCallbackEnabled Property

Syntax

SequenceFile.**UnloadCallbackEnabled**()

Data Type

Boolean

Purpose

This property controls whether TestStand calls the sequence file unload callback when TestStand unloads the sequence file. Set this property to False to prevent TestStand from calling the unload callback for the sequence file.

AsPropertyObject Method

Syntax

SequenceFile.**AsPropertyObject**()

Return Type

PropertyObject

Purpose

Returns the underlying PropertyObject that represents the SequenceFile object. Use the PropertyObject to modify, add, or remove custom properties of the object.

CreateCallbackOverrideSequence Method

Syntax

SequenceFile.**CreateCallbackOverrideSequence**(callbackName, allowCopyDefaultSteps)

Return Type

Sequence

Purpose

Use this method to create a callback sequence in the sequence file. The callback sequence you create overrides the model, engine, or front end callback of the same name as the callback you create. Refer to the *Callback Sequences* section in Chapter 1, *TestStand Architecture Overview*, for more information about callbacks.

Return Value

A reference to the sequence callback the method creates. Release this reference when you are finished using it.

Parameter	Type	Description
callbackName	String	The name of the callback sequence to override. Specify the sequence name of a model, engine, or front end callback. See DefaultModelCallbacks, and SeqFileCallbacks constants for possible values for this parameter.
allowCopyDefaultSteps	Boolean	Pass True to copy the steps of the sequence you are overriding into the new sequence.

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

DeleteSequence Method

Syntax

SequenceFile.**DeleteSequence**(index)

Purpose

Deletes the sequence you specify from the sequence file.

Parameter	Type	Description
index	Long	Pass a zero-based index to specify the sequence to delete.

GetModelSequenceFile Method

Syntax

SequenceFile.**GetModelSequenceFile**(modelDescriptionString)

Return Type

SequenceFile

Purpose

Returns a reference to the process model sequence file that TestStand associates with the sequence file on which you call the method. Release this reference when you are finished using it.

Remarks

Returns a NULL reference if the sequence file on which you call the method does not have a process model associated with it.

Parameter	Type	Description
modelDescriptionString	String	Returns a string that describes the process model file.

See Also

HasModel
Engine.GetStationModelSequenceFile

GetSequence Method

Syntax

SequenceFile.**GetSequence**(index)

Return Type

Sequence

Purpose

Gets a reference to a Sequence object that you specify by an index. Release your reference to this object when you are finished using it.

Parameter	Type	Description
index	Long	Pass a zero-based index to specify the sequence.

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

GetSequenceByName Method

Syntax

SequenceFile.**GetSequenceByName**(sequenceName)

Return Type

Sequence

Purpose

Gets a reference to a Sequence object that you specify by name. Release your reference to this object when you are finished using it.

Remarks

Returns an error if no Sequence exists with the name you specify.

Parameter	Type	Description
sequenceName	String	Pass the name of the sequence to which you want a reference. If the sequence name does not exist, then the method will report an error.

See Also

SequenceNameExists
GetSequenceIndex

GetSequenceIndex Method

Syntax

SequenceFile.**GetSequenceIndex**(sequenceName)

Return Type

Long

Purpose

Returns the index of the sequence in the sequence file that has the name you specify.

Return Value

Returns the index of the sequence whose name you specify. Returns -1 if no such sequence exists.

Parameter	Type	Description
sequenceName	String	Pass the name of a sequence in the sequence file.

See Also

SequenceNameExists
GetSequenceByName

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

IncChangeCount Method

Syntax

SequenceFile.**IncChangeCount**()

Purpose

When you make editing changes to a sequence file, call this funtion to indicate the change.

Remarks

All sequences which make editing changes to other sequences should use this method to indicate to the sequence editor or operator interface that changes have been made.

See Also

ChangeCount

InsertSequence Method

Syntax

SequenceFile.**InsertSequence**(sequenceToInsert)

Purpose

Inserts a sequence into the sequence file.

Remarks

Never insert a sequence into a sequence file when the sequence resides in another sequence file. The sequence reference that you pass must be the only reference to the sequence. You can obtain the sole reference to a sequence by calling Engine.NewSequence or SequenceFile.RemoveSequence.

Parameter	Type	Description
sequenceToInsert	Sequence	Pass the sequence to insert.

See Also

Engine.NewSequence
RemoveSequence

NewEditContext Method

Syntax

SequenceFile.**NewEditContext**()

Return Type

SequenceContext

Purpose

Returns a sequence context that approximates the sequence context TestStand creates when you run a sequence in the sequence file.

Remarks

You can pass the object this method returns as a parameter to the Engine.DisplayBrowseExprDialog method.

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

Return Value

A reference to a sequence context object. Release this reference when you are finished using it.

See Also

Engine.DisplayBrowseExprDialog

RemoveSequence Method

Syntax

SequenceFile.**RemoveSequence**(index)

Return Type

Sequence

Purpose

Removes a sequence from a sequence file and returns a reference to it. Release your reference to the sequence when you are finished using it.

Parameter	Type	Description
index	Long	Pass a zero-based index to specify the sequence to remove.

Save Method

Syntax

SequenceFile.**Save**(pathString)

Purpose

Saves the sequence file to disk.

Parameter	Type	Description
pathString	String	Pass the pathname with which to save the file. Pass an empty string to save the file using the same pathname with which TestStand last loaded or saved it.

SequenceNameExists Method

Syntax

SequenceFile.**SequenceNameExists**(sequenceName)

Return Type

Boolean

Purpose

Returns True if a sequence with the name you specify already exists in the sequence file.

Parameter	Type	Description
sequenceName	String	Specify a sequence name to search for.

See Also

GetSequenceByName
GetSequenceIndex

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

SequenceProperties Constants

Use these string constants to create lookupStrings when using the PropertyObject class to access the built-in properties of a sequence.

- Seq_CleanupProp
- Seq_MainProp
- Seq_SetupProp

See Also

PropertyObject

SequenceTypes Enumeration

This data type contains values that specify the type of a sequence. Use the values of this enumeration with the Sequence.Type property.

You can use the following constants with this data type.

- SeqType_Callback
- SeqType_CfgEntryPoint
- SeqType_ExeEntryPoint
- SeqType_Normal

See Also

Sequence
Sequence.Type

SpecifyModuleOptions Constants

These constants represent the options you can use with the Step.SpecifyModule method. Use the bitwise-OR operator to specify more than one option.

- SpecMod_AllowPrototypeChanges
- SpecMod_NoOptions
- SpecMod_NoSyntaxChecking
- SpecMod_ReadOnly

See Also

Step.SpecifyModule

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

Step

Objects of the Step class represent steps in TestStand sequences. In TestStand, a step can do many things, such as initializing an instrument, performing a complex test, or making a decision that affects the flow of execution in a sequence. You can obtain a reference to a step in a sequence by calling the Sequence.GetStep method.

Properties

AdapterKeyName
BreakOnStep
CanExecuteEditSubstep
CanSpecifyModule
CustomActionExpression
CustomFalseAction
CustomFalseActionTarget
CustomTrueAction
CustomTrueActionTarget
Description
FailAction
FailActionTarget
IconName
IsSequenceCall
LargeIcon
LargeIconIndex
LoopIncExpression
LoopInitExpression
LoopStatusExpression
LoopType
LoopWhileExpression
ModuleLoadOption
ModuleUnloadOption
Name
PassAction
PassActionTarget
PostExpression
Precondition
PreExpression
RecordResult
ResultStatus
RunMode
RunTimeRunMode
SmallIcon
SmallIconIndex
StatusExpression
StepFailCausesSequenceFail

Methods

AsPropertyObject
ExecuteEditSubstep
SpecifyModule

See Also

Sequence.GetStep
Sequence

AdapterKeyName Property (Read Only)

Syntax

Step.**AdapterKeyName**()

Data Type

String

Purpose

Returns the key name of the module adapter that the step uses.

Remarks

If the step does not use a module adapter, this property returns "None Adapter", the value of the constant AdapterKeyNames.NoneAdapterKeyName.

See Also

AdapterKeyNames

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

BreakOnStep Property

Syntax
Step.**BreakOnStep**()

Data Type
Boolean

Purpose

Set this property to True to cause TestStand to suspend execution before it executes the step.

CanExecuteEditSubstep Property (Read Only)

Syntax
Step.**CanExecuteEditSubstep**()

Data Type
Boolean

Purpose

If this property returns True, you can call the ExecuteEditSubstep method of the step.

See Also
ExecuteEditSubstep

CanSpecifyModule Property (Read Only)

Syntax
Step.**CanSpecifyModule**()

Data Type
Boolean

Purpose

If this property returns True, you can call the SpecifyModule method of the step.

See Also
SpecifyModule

CustomActionExpression Property

Syntax
Step.**CustomActionExpression**()

Data Type
String

Purpose

Use this property to get or set the custom post-action expression for the step.

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

See Also

CustomFalseAction
CustomFalseActionTarget
CustomTrueAction
CustomTrueActionTarget

CustomFalseAction Property

Syntax

Step.**CustomFalseAction**()

Data Type

String

Purpose

Use this property to get or set the type of action you want to occur when the custom post-action expression evaluates to False.

Remarks

Assign a PostActionValues string constant to the property to specify the type of Post action to perform.

See Also

CustomActionExpression
CustomFalseActionTarget
CustomTrueAction
CustomTrueActionTarget
PostActionValues

CustomFalseActionTarget Property

Syntax

Step.**CustomFalseActionTarget**()

Data Type

String

Purpose

Use this property to get or set the target for the Post action that the CustomFalseAction property specifies.

Remarks

If the CustomFalseAction is PostAction_GotoStep, the target is the name of the step. If the CustomFalseAction is PostAction_CallCallback, the target is the name of the callback sequence. For all other types of Post actions, the target property is not used.

See Also

CustomActionExpression
CustomFalseAction
CustomTrueAction
CustomTrueActionTarget

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

CustomTrueAction Property

Syntax

Step.**CustomTrueAction**()

Data Type

String

Purpose

Use this property to get or set the type of action you want to occur when the custom post-action expression evaluates to True.

Remarks

Assign a PostActionValues string constant to the property to specify the type of Post action to perform.

See Also

CustomActionExpression
CustomFalseAction
CustomFalseActionTarget
CustomTrueActionTarget
PostActionValues

CustomTrueActionTarget Property

Syntax

Step.**CustomTrueActionTarget**()

Data Type

String

Purpose

Use this property to get or set the target for the Post action that the CustomTrueAction property specifies.

Remarks

If the CustomTrueAction is PostAction_GotoStep, the target is the name of the step. If the CustomTrueAction is PostAction_CallCallback, the target is the name of the callback sequence. For all other types of Post actions, the target property is not used.

See Also

CustomActionExpression
CustomFalseAction
CustomFalseActionTarget
CustomTrueAction

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

Description Property (Read Only)

Syntax
Step.**Description**()

Data Type
String

Purpose
A string that describes the step.

Remarks
This string can change whenever you modify any of the step's settings.

FailAction Property

Syntax
Step.**FailAction**()

Data Type
String

Purpose
Use this property to get or set the type of Post action you want to occur if the step fails.

Remarks
Assign a PostActionValues string constant to the property to specify the type of Post action to perform.

See Also
PostActionValues
FailActionTarget

FailActionTarget Property

Syntax
Step.**FailActionTarget**()

Data Type
String

Purpose
Use this property to get or set the target for the Post action that the FailAction property specifies.

Remarks
If the FailAction is PostAction_GotoStep, the target is the name of the step. If the FailAction is PostAction_CallCallback, the target is the name of the callback sequence. For all other types of Post actions, the target property is not used.

See Also
FailAction

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

IconName Property (Read Only)

Syntax

Step.**IconName**()

Data Type

String

Purpose

Returns the filename of the icon for the step.

Remarks

If the step type for the step has no icon file, the property returns the filename of the icon for the module adapter that the step uses, and the SmallIcon and LargeIcon properties return the adapter icon.

Icon files are in the TestStand\Components\NI\Icons and TestStand\Components\User\Icons directories.

See Also

SmallIcon
LargeIcon
Adapter

IsSequenceCall Property (Read Only)

Syntax

Step.**IsSequenceCall**()

Data Type

Boolean

Purpose

Returns True if the step is a sequence call step.

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

LargeIcon Property (Read Only)

Syntax
Step.**LargeIcon**()

Data Type
Picture

Purpose
Returns the large version of the icon that represents the step.

Remarks
The steptype of the step defines the icon that the property returns. If the steptype does not specify an icon, the property returns the icon that the adapter of the step specifies.

See Also
LargeIconIndex
SmallIcon
SmallIconIndex
IconName

LargeIconIndex Property (Read Only)

Syntax
Step.**LargeIconIndex**()

Data Type
Long

Purpose
Returns a unique index for the large version of the icon that represents the step.

Remarks
The index is unique among all large icons that TestStand returns.

See Also
LargeIcon
SmallIcon
SmallIconIndex
IconName

LoopIncExpression Property

Syntax
Step.**LoopIncExpression**()

Data Type
String

Purpose
Use this property to get or set the loop increment expression for the step.

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

See Also
- LoopInitExpression
- LoopStatusExpression
- LoopType
- LoopWhileExpression

LoopInitExpression Property

Syntax
Step.**LoopInitExpression**()

Data Type
String

Purpose

Use this property to get or set the loop initialization expression for the step.

See Also
- LoopIncExpression
- LoopStatusExpression
- LoopType
- LoopWhileExpression

LoopStatusExpression Property

Syntax
Step.**LoopStatusExpression**()

Data Type
String

Purpose

Use this property to get or set the loop status result expression for the step.

See Also
- LoopIncExpression
- LoopInitExpression
- LoopType
- LoopWhileExpression

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

LoopType Property

Syntax

Step.**LoopType**()

Data Type

String

Purpose

Use this property to get or set the type of looping for the step.

Remarks

Use the StepLoopTypes constants to specify the value of the property.

See Also

LoopIncExpression
LoopInitExpression
LoopStatusExpression
LoopWhileExpression
StepLoopTypes

LoopWhileExpression Property

Syntax

Step.**LoopWhileExpression**()

Data Type

String

Purpose

Use this property to get or set the loop while expression for the step.

See Also

LoopIncExpression
LoopInitExpression
LoopStatusExpression
LoopType

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

ModuleLoadOption Property

Syntax

Step.**ModuleLoadOption**()

Data Type

ModuleLoadOptions

You can use the following constants with this data type.

- **LoadOption_DynamicLoad** -- Do not load the code module for a step until the step is ready to call it.
- **LoadOption_PreloadWhenExecuted** -- Load the code module for a step when any sequence in the sequence file containing the step begins executing.
- **LoadOption_PreloadWhenOpened** -- Load the code module for a step when TestStand loads into memory the sequence file containing the step.
- **LoadOption_UseStepLoadOption** -- Load each code module according to the load option for the step that uses it. This option is valid only for the SequenceFile.ModuleLoadOption property.

Purpose

Use this property to get or set the option that determines when TestStand loads the code module for the step.

Remarks

The SequenceFile.ModuleLoadOption property takes precedence over this property, unless the SequenceFile.ModuleLoadOption is set to LoadOption_UseStepLoadOption.

See Also

ModuleUnloadOption
SequenceFile.ModuleLoadOption

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

ModuleUnloadOption Property

Syntax

Step.**ModuleUnloadOption**()

Data Type

ModuleUnloadOptions

You can use the following constants with this data type.

- **UnloadOption_AfterSequenceExecution** -- Unload the code module for a step after the sequence containing the step finishes executing.
- **UnloadOption_AfterStepExecution** -- Unload the code module for a step after the step finishes executing.
- **UnloadOption_OnPreconditionFailure** -- Unload the code module for a step if the precondition for the step evaluates to False.
- **UnloadOption_UseStepUnloadOption** -- Unload the code module for a step according to the unload option for the step. This option is valid only for the SequenceFile.ModuleUnloadOption property.
- **UnloadOption_WithSequenceFile** -- Unload the code module for a step when TestStand unloads from memory the sequence file containing the step.

Purpose

Use this property to get or set the option that determines when TestStand unloads the code module for the step.

Remarks

The SequenceFile.ModuleUnloadOption property takes precedence over this property, unless the SequenceFile.ModuleUnloadOption is set to UnloadOption_UseStepUnloadOption.

See Also

ModuleLoadOption
SequenceFile.ModuleUnloadOption

Name Property

Syntax

Step.**Name**()

Data Type

String

Purpose

Use this property to get or set the name of the step.

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

PassAction Property

Syntax
Step.**PassAction**()

Data Type
String

Purpose
Use this property to get or set the type of post-action you want to occur if the step passes.

Remarks
Assign a PostActionValues string constant to the property to specify the type of post-action to perform.

See Also
PostActionValues
PassActionTarget

PassActionTarget Property

Syntax
Step.**PassActionTarget**()

Data Type
String

Purpose
Use this property to get or set the target for the Post action that the PassAction property specifies.

Remarks
If the PassAction is PostAction_GotoStep, the target is the name of the step. If the PassAction is PostAction_CallCallback, the target is the name of the callback sequence. For all other types of Post actions, the target property is not used.

See Also
PassAction

PostExpression Property

Syntax
Step.**PostExpression**()

Data Type
String

Purpose
Use this property to get or set the Post expression for the step.

Remarks
TestStand evaluates the Post expression after it calls the code module and Post Step substep for the step.

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

Precondition Property

Syntax

Step.**Precondition**()

Data Type

String

Purpose

Use this property to set or get the string that contains the preconditions for the step.

Remarks

This string is an expression with a Boolean result. Use the AnyOf() and AllOf() expression functions to specify more than one expression. The following is an example of an expression for a precondition that is based on the results of another step:

RunState.Sequence.Main["NameOfAnotherStep"].Result.Status == "Passed"

PreExpression Property

Syntax

Step.**PreExpression**()

Data Type

String

Purpose

Use this property to get or set the Pre expression for the step.

Remarks

TestStand evaluates the Pre expression before it calls the Pre Step substep and code module for the step.

RecordResult Property

Syntax

Step.**RecordResult**()

Data Type

Boolean

Purpose

Specifies whether to record the Result properties of the step. If set to True, TestStand records the result of the step unless the DisableResults property of the Engine is True or the Disable Results property of the sequence is True.

Remarks

This setting can be overridden by both the Sequence.DisableResults property and the Engine.DisableResults property.

See Also

Sequence.DisableResults
Engine.DisableResults

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

ResultStatus Property (Read Only)

Syntax

Step.**ResultStatus**()

Data Type

String

Purpose

Returns the status string for the step.

Remarks

Although you can define your own status strings, this property is usually set to one of the following values from the StepProperties constants:

ResultStatus_NoStatus, ResultStatus_Done, ResultStatus_Skipped, ResultStatus_Passed, ResultStatus_Failed, ResultStatus_Error, ResultStatus_Running, or ResultStatus_Looping

See Also

StepProperties

RunMode Property

Syntax

Step.**RunMode**()

Data Type

String

Purpose

Use this property to get or set the run mode of the step.

Remarks

The RunModes constants define the valid values for this property.

This property is the run mode that TestStand stores for the step in the sequence file. To set the run mode temporarily, set the RunTimeRunMode instead. Setting this property also sets the RunTimeRunMode. In the sequence editor, the sequence file window shows the RunMode property for the steps, and the execution window shows the RunTimeRunMode property for the steps.

See Also

RunModes
RunTimeRunMode

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

RunTimeRunMode Property

Syntax

Step.**RunTimeRunMode**()

Data Type

String

Purpose

Use this property to get or set the run-time run mode of the step.

Remarks

The RunModes constants define the valid values for this property.

This property is the run mode that TestStand uses when it executes the sequence file. When running a step, TestStand copies the value of the RunMode property to this property if you have not already set this property explicitly. Unlike the RunMode property, the value of this property is not saved but is instead discarded when the sequence file is unloaded. When you set this property, it has no effect on the RunMode property. In the sequence editor, the sequence file window shows the RunMode property for the steps, and the execution window shows the RunTimeRunMode property for the steps.

See Also

RunModes
RunMode

SmallIcon Property (Read Only)

Syntax

Step.**SmallIcon**()

Data Type

Picture

Purpose

Returns the small version of the icon that represents the step.

Remarks

The steptype of the step defines the icon that the property returns. If the steptype does not specify an icon, the property returns the icon that the adapter of the step specifies.

See Also

LargeIcon
LargeIconIndex
SmallIconIndex
IconName

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

SmallIconIndex Property (Read Only)

Syntax

Step.**SmallIconIndex**()

Data Type

Long

Purpose

Returns a unique index for the small version of the icon that represents the step.

Remarks

The index is unique among all small icons that TestStand returns.

See Also

LargeIcon
LargeIconIndex
SmallIcon
IconName

StatusExpression Property

Syntax

Step.**StatusExpression**()

Data Type

String

Purpose

Use this property to get or set the status expression for the step.

Remarks

Use this expression to set the ResultStatus property of the step. TestStand executes this expression after executing all other substeps and expressions for the step. The expression must evaluate to a string.

See Also

ResultStatus

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

StepFailCausesSequenceFail Property

Syntax

Step.**StepFailCausesSequenceFail**()

Data Type

Boolean

Purpose

Specifies whether failure of the step should cause the sequence to fail.

Remarks

If this property is set to True and the step fails, TestStand sets the internal status property of the sequence that contains the step to Failure. If the Sequence.GotoCleanupOnFailure property is True for the sequence, execution then jumps to the Cleanup step group of the sequence.

See Also

Sequence.GotoCleanupOnFailure

AsPropertyObject Method

Syntax

Step.**AsPropertyObject**()

Return Type

PropertyObject

Purpose

Returns the underlying PropertyObject that represents the Step object. Use the PropertyObject to modify, add, or remove custom properties of the object.

ExecuteEditSubstep Method

Syntax

Step.**ExecuteEditSubstep**()

Return Type

Boolean

Purpose

Executes the edit substep for the step, if one exists.

Remarks

Check the CanExecuteEditSubstep property before calling this method.

Return Value

Returns True if the edit substep modifies the step.

See Also

CanExecuteEditSubstep

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

SpecifyModule Method

Syntax

Step.**SpecifyModule**(specModOptions = SpecMod_NoOptions)

Return Type

Boolean

Purpose

Displays the Specify Module dialog box for the step if one exists.

Remarks

Check the CanSpecifyModule property to see if it is ok to call this method.

Return Value

Returns True if the Specify Module dialog box modifies the step.

Parameter	Type	Description
specModOptions	Long	Use one or more SpecifyModuleOptions to modify the behavior of the dialog box. This parameter has a default value of SpecMod_NoOptions.

See Also

CanSpecifyModule

StepGroups Enumeration

This data type contains values that specify the step group of a sequence.

You can use the following constants with this data type.

- StepGroup_Cleanup
- StepGroup_Main
- StepGroup_Setup

See Also

Sequence

StepLoopTypes Constants

These constants specify the valid values for the Step.LoopType property.

- LoopType_Custom
- LoopType_FixedNumLoops
- LoopType_NoLoop
- LoopType_PassFailCount

See Also

Step.LoopType

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

StepProperties Constants

Use these string constants to create lookupStrings to access properties of the built-in step types using the PropertyObject class. Notice that some of the constants refer to properties, whereas others refer to property values.

- Limits_HighProp
- Limits_LowProp
- Limits_StringProp
- NumMeasComp_EQ
- NumMeasComp_GE
- NumMeasComp_GELE
- NumMeasComp_GELT
- NumMeasComp_GT
- NumMeasComp_GTLE
- NumMeasComp_GTLT
- NumMeasComp_LE
- NumMeasComp_LOG
- NumMeasComp_LT
- NumMeasComp_NE
- NumMeasRadix
- Result_NumericProp
- Result_PassFailProp
- Result_StatusProp
- Result_StringProp
- ResultStatus_Done
- ResultStatus_Error
- ResultStatus_Failed
- ResultStatus_Looping
- ResultStatus_NoStatus
- ResultStatus_Passed
- ResultStatus_Running
- ResultStatus_Skipped
- Step_InBufProp
- Step_LimitsProp
- Step_MeasComparisonType
- Step_ResultProp
- Step_TSInfoProp
- StrMeasComp_CaseSensitive

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

- StrMeasComp_IgnoreCase
- TSInfo_StepAdditions

See Also

PropertyObject

StepTypes Constants

Constants containing the type names for the built-in step types. Use these constants with the Engine.NewStep method to create a step of a particular step type.

- StepType_Action
- StepType_CallExecutable
- StepType_Goto
- StepType_Label
- StepType_LimitLoader
- StepType_MessagePopup
- StepType_NumericMeasurement
- StepType_PassFailTest
- StepType_SequenceCall
- StepType_Statement
- StepType_StringMeasurement

See Also

Engine.NewStep

Thread

Threads are elements of an Execution. Each thread maintains a call stack that contains a SequenceContext object for each active sequence invocation. You can obtain a thread of an execution by calling the Execution.GetThread method.

Properties

CallStackSize	DisplayName	Execution
Id		

Methods

AsPropertyObject	ClearCurrentRTE	ClearTemporaryBreakpoint
DoInteractiveExecution	GetSequenceContext	PostUIMessage
SetStepInto	SetStepOut	SetStepOver

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

CallStackSize Property (Read Only)

Syntax

Thread.**CallStackSize**()

Data Type

Long

Purpose

Returns the number of call stack entries that currently exist for the thread.

Remarks

You can obtain a sequence context for each call stack entry by passing a call stack index to the GetSequenceContext method of this class.

See Also

GetSequenceContext

DisplayName Property (Read Only)

Syntax

Thread.**DisplayName**()

Data Type

String

Purpose

Returns the name to display for the thread.

Execution Property (Read Only)

Syntax

Thread.**Execution**()

Data Type

Execution

Purpose

Returns a reference to the Execution object to which the thread belongs.

Remarks

Release the Execution reference when you have finished using it.

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

Id Property (Read Only)

Syntax
Thread.**Id**()

Data Type
Long

Purpose
Returns a unique ID number for the thread. The ID number is unique with respect to all threads that you might initiate before you shut down the TestStand engine.

Remarks
Use this ID number to compare two Thread object references to determine whether or not they refer to the same underlying thread.

AsPropertyObject Method

Syntax
Thread.**AsPropertyObject**()

Return Type
PropertyObject

Purpose
Returns the underlying PropertyObject that represents the Thread object. Use the PropertyObject to modify, add, or remove custom properties of the object.

ClearCurrentRTE Method

Syntax
Thread.**ClearCurrentRTE**()

Purpose
Clears the current run-time error state of the thread.

Remarks
Use this method to cause the thread of execution to ignore a run-time error.

ClearTemporaryBreakpoint Method

Syntax
Thread.**ClearTemporaryBreakpoint**()

Purpose
Clears the temporary breakpoint you set by calling the SetStepInto, SetStepOut, or SetStepOver method of this class.

See Also
SetStepInto
SetStepOut
SetStepOver

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

DoInteractiveExecution Method

Syntax

Thread.**DoInteractiveExecution**(InteractiveArgsParam)

Purpose

Executes specific steps interactively.

Remarks

You can call this method only if the thread is currently suspended at a breakpoint. You can run steps only in the sequence and step group in which the execution is suspended

Parameter	Type	Description
InteractiveArgsParam	InteractiveArgs	Pass an InteractiveArgs object indicating the steps that are currently selected in the operator interface and that contain the looping information necessary for an interactive execution. The selected steps that you specify must be in the currently executing step group.

See Also

InteractiveArgs

GetSequenceContext Method

Syntax

Thread.**GetSequenceContext**(callStackIndex, frameId)

Return Type

SequenceContext

Purpose

Returns a reference to the SequenceContext object that corresponds to the call stack index you specify.

Parameter	Type	Description
callStackIndex	Long	A zero-based index into the call stack. You get the number of items on the call stack from the CallStackSize property. Call stack index 0 specifies the sequence context for the most recently executing step group.
frameId	Long	Returns a unique ID for the sequence context that this method returns. Use this ID with subsequent calls to GetSequenceContext to determine if execution is in the same sequence. This can help you minimize the number of items that you must update in the execution display in an operator interface.

See Also

CallStackSize

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

PostUIMessage Method

Syntax

Thread.**PostUIMessage**(eventCode, numericDataParam, stringDataParam, synchronous)

Purpose

Call this method to post a UIMessage to the current operator interface or sequence editor.

Remarks

The messages that a test developer might post from a step are UIMsg_ProgressPercent and UIMsg_ProgressText. These messages tell the operator interface to display a progress indicator or text message for the execution.

Parameter	Type	Description
eventCode	UIMessageCodes	Specifies the type of UIMessage.
numericDataParam	Double	Specifies numeric data to pass with the message. When you post a UIMsg_ProgressPercent event, this parameter specifies the percent done.
stringDataParam	String	Specifies string data to pass with the message. When you post a UIMsg_ProgressText event, this parameter specifies the text to display.
synchronous	Boolean	Pass True if you want the method to wait until the operator interface processes the message. Pass False if you want the method to return immediately.

See Also

UIMessage

SetStepInto Method

Syntax

Thread.**SetStepInto**()

Purpose

Call this method to set the execution to suspend again at the earliest possible point.

Remarks

The method does not resume the execution. The Execution.StepInto method, however, does the equivalent of calling this method on the foreground thread then resuming the execution.

See Also

Execution.StepInto
ClearTemporaryBreakpoint

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

SetStepOut Method

Syntax

Thread.**SetStepOut**()

Purpose

Call this method to set the execution to suspend again after execution of the current sequence completes.

Remarks

The method does not resume the execution. The Execution.StepOut method, however, does the equivalent of calling this method on the foreground thread then resuming the execution.

See Also

Execution.StepOut
ClearTemporaryBreakpoint

SetStepOver Method

Syntax

Thread.**SetStepOver**()

Purpose

Call this method to set the execution to suspend again after execution of the next step completes.

Remarks

The method does not resume the execution. The Execution.StepOver method, however, does the equivalent of calling this method on the foreground thread then resuming the execution.

See Also

Execution.StepOver
ClearTemporaryBreakpoint

TSError Enumeration

These are the error values that the TestStand ActiveX API can return. Whenever a method or property in the TestStand API fails, check the TSError code as follows:

In LabWindows/CVI:

If the HRESULT for an API function is equal to DISP_E_EXCEPTION, check the value of the sCode member of the ERRORINFO structure. If the HRESULT is not equal to DISP_E_EXCEPTION, then the error is not a TSError. You can use the value of the HRESULT to determine the type of error.

In LabVIEW:

Check the value of the code number in the "error out" cluster.

In Visual C/C++ MFC:

Check the value of the m_scError member variable of the COleDispatchException object.

In Visual Basic:

Check the value of Err.Number

You can use the following constants with this data type.

- TS_Err_AccessDenied
- TS_Err_ArrayIndexOutOfBounds

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

- TS_Err_ArrayLocked
- TS_Err_AutomationObjNotValid
- TS_Err_BadExpressionError
- TS_Err_BadFileFormat
- TS_Err_BadNetPath
- TS_Err_BadPropertyOrVariableName
- TS_Err_CurrentSeqFileNotAvailable
- TS_Err_CVIAutoCmdFailed
- TS_Err_CVICantConnectToTecrunServer
- TS_Err_CVIFuncNotFoundInModule
- TS_Err_CVIModuleHasUnresolvedReferences
- TS_Err_CVINotReg
- TS_Err_CVIOleError
- TS_Err_CVIRegGenericReadError
- TS_Err_CVIRegKeyNotFound
- TS_Err_CVIRegValueNotFound
- TS_Err_CVIRegValueTypeMismatch
- TS_Err_CVIUnableToTerminateUserProgInCVI
- TS_Err_DDEFail
- TS_Err_DiskFull
- TS_Err_DispMissingParamID
- TS_Err_DispMissingParamName
- TS_Err_DispMissingRequiredArg
- TS_Err_DispUnknownInterface
- TS_Err_DispUnknownMemberID
- TS_Err_DispUnknownMemberName
- TS_Err_DispUnknownParamID
- TS_Err_DispUnknownParamName
- TS_Err_DispWrongNumPositionalParams
- TS_Err_DLLNotLoadable
- TS_Err_DriveNotReady
- TS_Err_DuplicateItemOrValue
- TS_Err_EvaluationContextNotAvailable
- TS_Err_ExprTypeIncompatibleWithParameter
- TS_Err_ExprValueNotSuperSetOfParameter
- TS_Err_FailToRegisterClipFormat

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

- TS_Err_FileAlreadyExists
- TS_Err_FileFormatIsOutOfDate
- TS_Err_FileNotConvertableToSeqFile
- TS_Err_FileWasNotFound
- TS_Err_FunctionNotFoundInLib
- TS_Err_IllegalOperationOnValue
- TS_Err_IncompatibleParameters
- TS_Err_IndexOutOfRange
- TS_Err_InvalidAdapterName
- TS_Err_InvalidDrive
- TS_Err_InvalidPathname
- TS_Err_IOError
- TS_Err_ItemCannotBeDeleted
- TS_Err_LValueExpected
- TS_Err_LVAutoServerError
- TS_Err_MismatchedArrayBounds
- TS_Err_MissingType
- TS_Err_ModuleLoadFailure
- TS_Err_ModuleNotSpecified
- TS_Err_NameAlreadyInUse
- TS_Err_NoError
- TS_Err_NoFileAssoc
- TS_Err_NoItemsInList
- TS_Err_ObjectCannotBeAdded
- TS_Err_ObjectTypeIncompatibleWithParameter
- TS_Err_OperationCanceled
- TS_Err_OperationFailed
- TS_Err_OperationInProgress
- TS_Err_OperationOnlyValidWhenSuspended
- TS_Err_OperationTimedOut
- TS_Err_OS_Exception
- TS_Err_OutOfMemory
- TS_Err_PathNotFound
- TS_Err_ProgramError
- TS_Err_RegistryAccessError
- TS_Err_RegistryItemNotFound

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

- TS_Err_RStringNotFound
- TS_Err_SequenceAborted
- TS_Err_SequenceTerminated
- TS_Err_SharingViolation
- TS_Err_SingleDimensionalNumericArrayExpected
- TS_Err_StepTypeNotFound
- TS_Err_ThreadCreationFailed
- TS_Err_TooManyItems
- TS_Err_TypeCannotBeDeleted
- TS_Err_TypeConflict
- TS_Err_TypeMismatchError
- TS_Err_TypeWithDependingInstancesCannotBeDeleted
- TS_Err_UnableToAllocateSystemResource
- TS_Err_UnableToCloseFile
- TS_Err_UnableToInitializeOLESystemDLLs
- TS_Err_UnableToLaunchCVI
- TS_Err_UnableToOpenFile
- TS_Err_UnableToPassByReference
- TS_Err_UnexpectedSystemError
- TS_Err_UnexpectedType
- TS_Err_UnknownFunctionOrSequenceName
- TS_Err_UnknownType
- TS_Err_UnknownVariableOrProperty
- TS_Err_UnRecognizedValue
- TS_Err_ValueIsInvalidOrOutOfRange
- TS_Err_WriteProtected
- TS_Err_WrongNumberOfArrayIndices
- TS_Err_WrongNumberOfParameters

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

UIMessage

TestStand uses UIMessage objects to pass information to the operator interface or sequence editor about the state of the engine and the current executions. You can obtain UIMessage objects in an operator interface program by providing a callback to the engine. The engine calls the callback when it has a UIMessage object to pass. If you do not provide a callback, you can set the Engine.UIMessagePollingEnabled property to True and call the Engine.GetUIMessage method to poll for UIMessage objects.

Properties

Event	Execution	IsSynchronous
NumericData	StringData	Thread

Methods

AsPropertyObject

Event Property (Read Only)

Syntax

UIMessage.**Event**()

Data Type

UIMessageCodes

You can use the following constants with this data type.

- **UIMsg_AbortingExecution** -- TestStand sends this event just before aborting an execution. TestStand does not actually abort until execution until the user interface releases the UIMessage.
- **UIMsg_BreakOnBreakpoint** -- TestStand sends this event to the user interface to notify it that the execution has suspended. The user interface must update its display accordingly.
- **UIMsg_BreakOnRunTimeError** -- TestStand sends this event to the user interface to notify it that a runtime error has been encountered in the execution. The user interface must update its display accordingly.
- **UIMsg_BreakOnUserRequest** -- TestStand sends this event to the user interface to notify it that the execution suspended in response to a user request. The user interface must update its display accordingly.
- **UIMsg_EndExecution** -- TestStand sends this event after an execution completes. The user interface must update its display accordingly.
- **UIMsg_EndFileExecution** -- TestStand sends this event when the execution finishes using a sequence file. TestStand specifies the file in the UIMessage.
- **UIMsg_EndInteractiveExecution** -- TestStand sends this event whenever an interactive execution ends.
- **UIMsg_KillingExecutionThreads** -- TestStand sends this event just before killing the threads in an execution. TestStand does not actually kill the threads until the user interface releases the UIMessage.
- **UIMsg_ProgressPercent** -- Tests in TestStand post this message to the user interface to notify it to update its progress indicator.
- **UIMsg_ProgressText** -- TestStand step modules post this message to the user interface to notify it to update its progress message.
- **UIMsg_ResumeFromBreak** -- TestStand sends this event when an execution resumes from being suspended at a breakpoint. The execution resumes when the UIMessage is released.
- **UIMsg_ShutDownCancelled** -- TestStand send this message to notify the user interface that the user cancelled a pending shutdown.
- **UIMsg_ShutDownComplete** -- TestStand sends this message when a shutdown completes. If this is not the final shutdown, the user interface can continue to load and execute sequences.

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

- **UIMsg_StartExecution** -- TestStand sends this event when an execution begins to notify the user interface to update its display accordingly.
- **UIMsg_StartFileExecution** -- TestStand sends this event when it begins using a sequence file. TestStand specifies the file in the UIMessage.
- **UIMsg_StartInteractiveExecution** -- TestStand sends this event whenever an interactive execution begins.
- **UIMsg_TerminatingExecution** -- TestStand sends this event just before terminating an execution. The execution does not actually terminate until the user interface releases the UIMessage.
- **UIMsg_TerminatingInteractiveExecution** -- TestStand sends this event just before terminating an interactive execution. The interactive execution does not actually terminate until the UIMessage is released.
- **UIMsg_TerminationCancelled** -- TestStand sends this event to notify the user interface to update its display when a pending termination is cancelled.
- **UIMsg_Trace** -- TestStand sends this event to the user interface to notify it that the execution has reached a trace point. The user interface must update its display accordingly.
- **UIMsg_UserMessageBase** -- All user-defined error codes must be greater than or equal to the value of this constant.

Purpose

Returns the event code that describes the type of UIMessage.

Execution Property (Read Only)

Syntax

UIMessage.**Execution**()

Data Type

Execution

Purpose

Returns a reference to the execution that the event applies to.

Remarks

Release the Execution reference when you are done using it. If the event is not associated with an execution, this property returns a NULL reference.

See Also

Execution

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

IsSynchronous Property (Read Only)

Syntax

UIMessage.**IsSynchronous**()

Data Type

Boolean

Purpose

Returns True if the thread that posted the event is blocked waiting for you to release the UIMessage.

Remarks

Threads can post UIMessages synchronously or asynchronously. If a thread posts a UIMessage synchronously, the thread blocks until the last reference to the UIMessage is released.

See Also

Thread.PostUIMessage

NumericData Property (Read Only)

Syntax

UIMessage.**NumericData**()

Data Type

Double

Purpose

Returns the numeric value that the thread that posted the UIMessage passed to Thread.PostUIMessage.

Remarks

Currently, the only built-in UIMessage event that uses numeric data is the UIMsg_ProgressPercent event. If you define your own UIMessage events, you can pass numeric data to Thread.PostUIMessage.

See Also

Thread.PostUIMessage

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

StringData Property (Read Only)

Syntax

UIMessage.**StringData**()

Data Type

String

Purpose

Returns the string value that the thread that posted the UIMessage passed to Thread.PostUIMessage.

Remarks

Currently, the only built-in UIMessage event that uses string data is the UIMsg_ProgressText event. If you define your own UIMessage events, you can pass string data to Thread.PostUIMessage.

See Also

Thread.PostUIMessage

Thread Property (Read Only)

Syntax

UIMessage.**Thread**()

Data Type

Thread

Purpose

Returns a reference to the thread that the event applies to.

Remarks

Release the Thread reference when you are done using it. If the event is not associated with a thread, this property returns a NULL reference.

See Also

Thread

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

AsPropertyObject Method

Syntax

UIMessage.**AsPropertyObject**()

Return Type

PropertyObject

Purpose

Returns the underlying PropertyObject that represents the UIMessage object. Use the PropertyObject to modify, add, or remove custom properties of the object.

UIMessageCodes Enumeration

These constants specify the type of event that a particular UIMessage object represents.

You can use the following constants with this data type.

- **UIMsg_AbortingExecution** -- TestStand sends this event just before aborting an execution. TestStand does not actually abort execution until the user interface releases the UIMessage.
- **UIMsg_BreakOnBreakpoint** -- TestStand sends this event to the user interface to notify it that the execution has suspended. The user interface must update its display accordingly.
- **UIMsg_BreakOnRunTimeError** -- TestStand sends this event to the user interface to notify it that a run-time error has been encountered in the execution. The user interface must update its display accordingly.
- **UIMsg_BreakOnUserRequest** -- TestStand sends this event to the user interface to notify it that the execution suspended in response to a user request. The user interface must update its display accordingly.
- **UIMsg_EndExecution** -- TestStand sends this event after an execution completes. The user interface must update its display accordingly.
- **UIMsg_EndFileExecution** -- TestStand sends this event when the execution finishes using a sequence file. TestStand specifies the file in the UIMessage.
- **UIMsg_EndInteractiveExecution** -- TestStand sends this event whenever an interactive execution ends.
- **UIMsg_KillingExecutionThreads** -- TestStand sends this event just before killing the threads in an execution. TestStand does not actually kill the threads until the user interface releases the UIMessage.
- **UIMsg_ProgressPercent** -- Tests in TestStand post this message to the user interface to notify it to update its progress indicator.
- **UIMsg_ProgressText** -- TestStand step modules post this message to the user interface to notify it to update its progress message.
- **UIMsg_ResumeFromBreak** -- TestStand sends this event when an execution resumes after being suspended at a breakpoint. The execution resumes when the UIMessage is released.
- **UIMsg_ShutDownCancelled** -- TestStand sends this message to notify the user interface that the user cancelled a pending shutdown.
- **UIMsg_ShutDownComplete** -- TestStand sends this message when a shutdown completes. If this is not the final shutdown, the user interface can continue to load and execute sequences.
- **UIMsg_StartExecution** -- TestStand sends this event when an execution begins. This event serves to notify the user interface to update its display accordingly.
- **UIMsg_StartFileExecution** -- TestStand sends this event when it begins using a sequence file. TestStand specifies the file in the UIMessage.
- **UIMsg_StartInteractiveExecution** -- TestStand sends this event whenever an interactive execution begins.

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

- **UIMsg_TerminatingExecution** -- TestStand sends this event just before terminating an execution. The execution does not actually terminate until the user interface releases the UIMessage.
- **UIMsg_TerminatingInteractiveExecution** -- TestStand sends this event just before terminating an interactive execution. The interactive execution does not actually terminate until the UIMessage is released.
- **UIMsg_TerminationCancelled** -- TestStand sends this event to notify the user interface to update its display when a pending termination is cancelled.
- **UIMsg_Trace** -- TestStand sends this event to the user interface to notify it that the execution has reached a trace point. The user interface must update its display accordingly.
- **UIMsg_UserMessageBase** -- All user-defined error codes must be greater than or equal to the value of this constant.

See Also

UIMessage
UIMessage.Event

User

Objects of the User class represent the data and privileges that TestStand associates with a TestStand user. Call the Engine.GetUser or Engine.NewUser method to obtain a User object for a particular user. Obtain the object for the user that is currently logged in from the Engine.CurrentUser property.

Properties

FullName	LoginName	Password
Privileges		

Methods

AsPropertyObject	HasPrivilege	ValidatePassword

See Also

Engine.GetUser
Engine.NewUser
Engine.CurrentUser

FullName Property

Syntax

User.**FullName**()

Data Type

String

Purpose

The full name of the user.

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

LoginName Property

Syntax
User.**LoginName**()

Data Type
String

Purpose
The login name of the user.

Password Property

Syntax
User.**Password**()

Data Type
String

Purpose
The password of the user.

Remarks
When you set this property, TestStand scrambles the password before storing it internally. When you get this property, TestStand returns the scrambled password to you. Returning the scrambled password prevents users from accessing passwords programmatically without authorization.

Privileges Property (Read Only)

Syntax
User.**Privileges**()

Data Type
PropertyObject

Purpose
Returns the PropertyObject that contains the privilege settings for the user.

AsPropertyObject Method

Syntax
User.**AsPropertyObject**()

Return Type
PropertyObject

Purpose
Returns the underlying PropertyObject that represents the User object. Use the PropertyObject to modify, add, or remove custom properties of the object.

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

HasPrivilege Method

Syntax

User.**HasPrivilege**(privilegeName)

Return Type

Boolean

Purpose

Returns True if the user has the privilege you specify by name.

Parameter	Type	Description
privilegeName	String	Specifies a privilege to check by name. You can specify the name of any privilege property in the user privileges property tree, including a group privilege. You do not have to include the names of all the containing groups. If you specify only the base name of a privilege and more than one group contains a privilege of that name, the method returns the value of the first privilege it finds with that name.

See Also

UserPrivileges

ValidatePassword Method

Syntax

User.**ValidatePassword**(passwordString)

Return Type

Boolean

Purpose

Returns True if the string you specify matches the user's password.

Parameter	Type	Description
passwordString	String	The string to compare with the user's password.

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

UserPrivileges Constants

Use these string constants to check whether a user has a particular built-in privilege. Use the Engine.CurrentUserHasPrivilege or User.HasPrivilege method with these constants.

- Priv_Abort
- Priv_ConfigAdapter
- Priv_ConfigApp
- Priv_ConfigEngine
- Priv_CtrlExecFlow
- Priv_EditSequenceFiles
- Priv_EditStationGlobals
- Priv_EditTypes
- Priv_EditUsers
- Priv_Execute
- Priv_LoopSelectedTests
- Priv_RunAnySequence
- Priv_RunSelectedTests
- Priv_SinglePass
- Priv_Terminate

See Also

Engine.CurrentUserHasPrivilege
User.HasPrivilege

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

Data Types

Visual Basic	C/C++	Description
Boolean	BOOL	Has the value True (-1) or False (0).
Color	OLECOLOR	A color. In many containers, colors are treated as 32-bit integers.
Double	double	64-bit floating point number.
Double Array		An array of doubles.
Font	LPFONTDISP	An OLE Automation Interface to a font.
Integer	short	16-bit signed integer.
Long	long	32-bit signed integer.
LPUNKNOWN	LPUNKNOWN	
Object	LPDISPATCH	A generic OLE Automation Interface.
OLE_XPOS_PIXELS	long	
OLE_YPOS_PIXELS	long	
Picture	LPPICTUREDISP	An OLE Automation Interface to a picture or image.
Single	single	32-bit floating point number.
String	BSTR	A string.
Variant	LPVARIANT	A variant.
Void	void	Nothing.
Window	OLE_HANDLE	A handle to a window.

National Instruments Technical Support

National Instruments has technical assistance through electronic, fax, and telephone systems to quickly provide the information you need. Our electronic services include a World Wide Web site, an FTP site, a fax-on-demand system, and e-mail support. If you have a hardware or software problem, first try the electronic support systems. If the information available on these systems does not answer your questions, we offer fax and telephone support through our technical support centers, which are staffed by applications engineers.

You can contact National Instruments using one of the following methods:

- WWW support
- FTP support
- FaxBack support
- E-mail support (currently U.S. only)
- Telephone and fax support

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.



About This File

Part Number: 322019B-01, Edition Date: January 1999

Copyright

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved. Under the copyright laws, this publication may not be reproduced or transmitted in any form, electronic or mechanical, including photocopying, recording, storing in an information retrieval system or translating, in whole or in part, without the prior written consent of National Instruments Corporation.

For more information, see the Warranty, Trademarks, Medical Warning and Technical Support topics.

Warranty, Trademarks, Medical Warning

Limited Warranty
Trademarks
Medical Warning

FTP Support

To access our FTP site, log on to our Internet host, `ftp://ftp.natinst.com` as anonymous and use your e-mail address, such as `joesmith@anywhere.com`, as your password. The support files and documents are located in the `/support` directories.

WWW Support

For information about National Instruments, point your Web browser to `www.natinst.com`.

FaxBack Support

FaxBack is a 24-hour information retrieval system containing a library of documents on a wide range of technical information. You can access **FaxBack** from a touch-tone telephone at the following number:

(512) 418-1111

E-mail Support (Currently U.S. Only)

You can submit technical support questions to the National Instruments engineering team through e-mail at the Internet address listed below. Remember to include your name, address, and phone number so we can contact you with solutions and suggestions.
`support@natinst.com`

Telephone and Fax Support

National Instruments has branch offices all over the world. Use the list below to find the technical support number for your country. If there is no National Instruments office in your country, contact the source from which you purchased your software to obtain support.

Country	Telephone	Fax
Australia	03 9879 5166	03 9879 6277
Austria	0662 45 79 90 0	0662 45 79 90 19
Belgium	02 757 00 20	02 757 03 11
Brazil	011 288 3336	011 288 8528
Canada (Ontario)	905 785 0085	905 785 0086
Canada (Québec)	514 694 8521	514 694 4399
Denmark	45 76 26 00	45 76 26 02
Finland	09 725 725 11	09 725 725 55
France	01 48 14 24 24	01 48 14 24 14
Germany	089 741 31 30	089 714 60 35
Hong Kong	2645 3186	2686 8505
Israel	03 6120092	03 6120095
Italy	02 413091	02 41309215
Japan	03 5472 2970	03 5472 2977
Korea	02 596 7456	02 596 7455
Mexico	5 520 2635	5 520 3282
Netherlands	0348 433466	0348 430673
Norway	32 84 84 00	32 84 86 00
Singapore	2265886	2265887
Spain	91 640 0085	91 640 0533
Sweden	08 730 49 70	08 730 43 70
Switzerland	056 200 51 51	056 200 51 55
Taiwan	02 377 1200	02 737 4644
United Kingdom	01635 523545	01635 523154
United States	512 795 8248	512 794 5678

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.

Limited Warranty

The media on which you receive National Instruments software are warranted not to fail to execute programming instructions, due to defects in materials and workmanship, for a period of 90 days from date of shipment, as evidenced by receipts or other documentation. National Instruments will, at its option, repair or replace software media that do not execute programming instructions if National Instruments receives notice of such defects during the warranty period. National Instruments does not warrant that the operation of the software shall be uninterrupted or error free.

A Return Material Authorization (RMA) number must be obtained from the factory and clearly marked on the outside of the package before any equipment will be accepted for warranty work. National Instruments will pay the shipping costs of returning to the owner parts which are covered by warranty.

National Instruments believes that the information in this manual is accurate. The document has been carefully reviewed for technical accuracy. In the event that technical or typographical errors exist, National Instruments reserves the right to make changes to subsequent editions of this document without prior notice to holders of this edition. The reader should consult National Instruments if errors are suspected. In no event shall National Instruments be liable for any damages arising out of or related to this document or the information contained in it.

Except as specified herein, National Instruments makes no warranties, express or implied, and specifically disclaims any warranty of merchant ability or fitness for a particular purpose. Customer's right to recover damages caused by fault or negligence on the part of National Instruments shall be limited to the amount theretofore paid by the customer. National Instruments will not be liable for damages resulting from loss of data, profits, use of products, or incidental or consequential damages, even if advised of the possibility thereof. This limitation of the liability of National Instruments will apply regardless of the form of action, whether in contract or tort, including negligence. Any action against National Instruments must be brought within one year after the cause of action accrues. National Instruments shall not be liable for any delay in performance due to causes beyond its reasonable control. The warranty provided herein does not cover damages, defects, malfunctions, or service failures caused by owner's failure to follow the National Instruments installation, operation, or maintenance instructions; owner's modification of the product; owner's abuse, misuse, or negligent acts; and power failure or surges, fire, flood, accident, actions of third parties, or other events outside reasonable control.

Trademarks

CVI™, LabVIEW™, `natinst.com`™, National Instruments™, the National Instruments logo, and TestStand™ are trademarks of National Instruments Corporation.

Product and company names listed are trademarks or trade names of their respective companies.

WARNING REGARDING MEDICAL AND CLINICAL USE OF NATIONAL INSTRUMENTS PRODUCTS

National Instruments products are not designed with components and testing intended to ensure a level of reliability suitable for use in treatment and diagnosis of humans. Applications of National Instruments products involving medical or clinical treatment can create a potential for accidental injury caused by product failure, or by errors on the part of the user or application designer. Any use or application of National Instruments products for or involving medical or clinical treatment must be performed by properly trained and qualified medical personnel, and all traditional medical safeguards, equipment, and procedures that are appropriate in the particular situation to prevent serious injury or death should always continue to be used when National Instruments products are being used. National Instruments products are NOT intended to be a substitute for any form of established process, procedure, or equipment used to monitor or safeguard human health and safety in medical or clinical treatment.

© Copyright 1998, 1999, National Instruments Corporation. All rights reserved.