

## **User Interface Component Overview**

---

You use the User Interface component to add user interface controls to your application. You can modify the User Interface controls programmatically or by using the property pages in Visual C++. Also, you can bind User Interface controls to a DataSocket server, thus allowing the controls to read from and write to the server.

### **Top-Level Classes**

---

[CNiButton](#) - allows you to add buttons to your user interface.

[CNiGraph](#) - allows you to add graphs to your user interface.

[CNiKnob](#) - allows you to add knobs to your user interface.

[CNiNumEdit](#) - allows you to add NumEdit controls to your user interface.

[CNiSlide](#) - allows you to add slides to your user interface.

#### **Notes:**

- You can use the CNiAxis class with the CNiKnob and CNiSlide classes to interface to a single axis of the knob or slide control. This allows you to modify the appearance and behavior of the axis.
- Because the User Interface component includes ActiveX controls that link to the MFC DLL, projects that you design to use Measurement Studio User Interface controls cannot link to static MFC.
- The User Interface component includes classes that reference ActiveX objects. Although these classes provide support to allow you access the included ActiveX objects from any thread, you must initialize COM in each thread that accesses the User Interface classes. To initialize COM in a thread, call the system function `::CoInitialize` or the system function `::CoInitializeEx` from within the thread.

## User Interface Example Programs

---

This topic includes summaries of and links to the example programs associated with the User Interface component.

Advanced Button	The Advanced Button example demonstrates how to use the advanced features of a Measurement Studio Button control.	<a href="#">Load example in VC++</a>	<a href="#">Run example</a>
Simple Button	The Simple Button example demonstrates the various types of button styles that are available with the Measurement Studio button control.	<a href="#">Load example in VC++</a>	<a href="#">Run example</a>
Annotations	The Annotations example demonstrates how to use the annotations feature of the Measurement Studio graph control.	<a href="#">Load example in VC++</a>	<a href="#">Run example</a>
Axes	The Axes example demonstrates how to manually set the minimum and maximum values on graph axes and how to use autoscaling for graph axes.	<a href="#">Load example in VC++</a>	<a href="#">Run example</a>
Cursors	The Cursors example demonstrates how to set and get the position of a graph cursor and how to use cursor snap modes.	<a href="#">Load example in VC++</a>	<a href="#">Run example</a>
Multiple Cursors	The Multiple Cursors example demonstrates how to use multiple cursors and shows the snap mode options for the cursors.	<a href="#">Load example in VC++</a>	<a href="#">Run example</a>
Multiple Y Axis	The Multiple Y Axis example demonstrates how you can have multiple Y axes.	<a href="#">Load example in VC++</a>	<a href="#">Run example</a>
Plots VS Charts	The Plots VS Charts example demonstrates the difference in the plotting routines for graphs and stripcharts.	<a href="#">Load example in VC++</a>	<a href="#">Run example</a>

Graph Example	The Graph Example example generates data plots based on the number of traces, the number of points per trace, and the specification for stacking the plots.	<a href="#">Load example in VC++</a>	<a href="#">Run example</a>
Simple Graph	The Simple Graph example demonstrates some of the basic features of a graph control.	<a href="#">Load example in VC++</a>	<a href="#">Run example</a>
Graph Benchmark	The Graph Benchmark example demonstrates the performance of CNiGraph with various test parameters including plot style ( plotting or charting ), multiple plots versus a single plot, and parameters that determine if the data is plotted to the graph or directly to the plot objects of the graph.	<a href="#">Load example in VC++</a>	<a href="#">Run example</a>
Knob Styles	The Knob Styles example demonstrates the various styles of the knob control.	<a href="#">Load example in VC++</a>	<a href="#">Run example</a>
Simple Knob	The Simple Knob example demonstrates the various styles of the knob control.	<a href="#">Load example in VC++</a>	<a href="#">Run example</a>
Range Checking	The Range Checking example demonstrates the range checking feature of the numedit control.	<a href="#">Load example in VC++</a>	<a href="#">Run example</a>
Simple Numeric	The Simple Numeric example demonstrates many styles of the numedit control.	<a href="#">Load example in VC++</a>	<a href="#">Run example</a>
Advanced Slide	The Advanced Slide example demonstrates how to use features of the Measurement Studio for Visual C++ slide controls to create custom slide controls.	<a href="#">Load example in VC++</a>	<a href="#">Run example</a>
Simple	The Simple Slide example demonstrates how to	<a href="#">Load example</a>	<a href="#">Run</a>

Slide	use the basic functionality of Measurement Studio for Visual C++ slide controls.	<a href="#">in VC++</a>	<a href="#">example</a>
Slide Styles	The Slide Styles example demonstrates different styles for slide controls.	<a href="#">Load example in VC++</a>	<a href="#">Run example</a>

# **CNiAnnotation**



**Class**

Declared in:  
**NiAnnotation.h**





## Overview

---

`CNiAnnotation` encapsulates the interface to a single annotation on a graph control, which allows you to modify the annotation's appearance and behavior.

You obtain individual annotations using the `Annotations` property on a `CNiGraph` object.

Note that you must initialize a `CNiAnnotation` object from an existing object. If you do not initialize a `CNiAnnotation` object from an existing object, a `CNiObjectNotInUsableState` exception will be thrown when you attempt to manipulate the instance of the `CNiAnnotation`.

 [Hierarchy Chart](#)

## ▲ Base Classes

---

### ▲ CNiInterface

## ◆ Data Items

---

◆ <a href="#">CNIArrow</a>	<b><u>Arrow</u></b>	Returns the annotation arrow, which is a CNIArrow object.
◆ <a href="#">CNICaption</a>	<b><u>Caption</u></b>	Returns the annotation caption, which is a CNICaption object.
◆ CoordinateTypes	<b><u>CoordinateType</u></b>	Specifies the coordinate system the annotation uses.
◆ <a href="#">bool</a>	<b><u>Enabled</u></b>	Specifies if the annotation generates mouse events or if you can drag the annotation in annotation tracking mode.
◆ <a href="#">CString</a>	<b><u>Name</u></b>	Specifies the annotation name.
◆ <a href="#">CNIPlot</a>	<b><u>Plot</u></b>	Specifies the plot associated with the annotation.
◆ <a href="#">long</a>	<b><u>PointIndex</u></b>	Specifies the point associated with the annotation on the plot.
◆ <a href="#">CNIShape</a>	<b><u>Shape</u></b>	Returns the annotation shape, which is a CNIShape object.
◆ SnapModes	<b><u>SnapMode</u></b>	Specifies the snap mode of the annotation shape.
◆ <a href="#">bool</a>	<b><u>Visible</u></b>	Specifies if the annotation is visible or hidden.

## ◆ Constructors

---

- ◆ **CNiAnnotation()** Default constructor.
- ◆ **CNiAnnotation**( CWAnnotation\_CI\* pCustom, [CNiInterface::ThreadAccess](#) option ) Constructor that attaches to the specified CWAnnotation\_CI pointer.
- ◆ **CNiAnnotation**( [const](#) CNiAnnotation& source ) Copy constructor.

## ◆ **D e s t r u c t o r s**

---

◆ `~CNIAnnotation()`

Destructor.

## ◆ **F u n c t i o n s**

---

- ◆ `static const IID &` **GetId()** Returns the globally unique identifier (GUID) of the ActiveX interface to which this class connects.
  
- ◆ `const CNiAnnotation &` **`operator =( const CNiAnnotation& source )`** Assignment operator.
  
- ◆ `void` **SetBuiltinStyle( AnnotationStyles Style )** Sets the annotation properties to represent the style specified.

## Example

---

```
// Change the visibility of the first annotation on the graph.
```

```
CniGraph graph;
```

```
CniAnnotation annotation = graph.Annotations.Item(1);
```

```
annotation.Visible = false;
```

---

# **CNiAnnotations**



**Class**

Declared in:  
**NiAnnotations.h**





## Overview

---

CNiAnnotations encapsulates the interface to the Annotations property of a CNiGraph object, which allows you to access and remove annotations associated with the graph control.

 [Hierarchy Chart](#)

## ▲ Base Classes

---

### ▲ CNiInterface

## ◆ Data Items

---

◆ [short](#)      [Count](#) Returns the number of annotations in the collection.

## ◆ Constructors

---

- ◆ **CNiAnnotations()** Default constructor.
- ◆ **CNiAnnotations**( CWAnnotations\_CI\* pCustom, [CNiInterface::ThreadAccess](#) option ) Constructor that attaches to the specified CWAnnotations\_CI pointer.
- ◆ **CNiAnnotations**( [const](#) CNiAnnotations& source ) Copy constructor.

## ◆ **D e s t r u c t o r s**

---

◆ `~CNIAnnotations()`

Destructor.

## ◆ **F u n c t i o n s**

---

◆ <u>CNiAnnotation</u>	<u>Add()</u>	Adds an annotation to the collection and returns the new annotation.
◆ static const IID &	<u>GetId()</u>	Returns the globally unique identifier (GUID) of the ActiveX interface to which this class connects.
◆ <u>CNiAnnotation</u>	<u>Item</u> ( long annotationIndex )	Returns the specified annotation from the collection.
◆ <u>CNiAnnotation</u>	<u>Item</u> ( const <u>CString</u> & annotationName )	Returns the specified annotation from the collection.
◆ const CNiAnnotations &	operator =( const CNiAnnotations& source )	Assignment operator.
◆ void	<u>Remove</u> ( long annotationIndex )	Removes the specified annotation from the collection.
◆ void	<u>Remove</u> ( const <u>CString</u> & annotationName )	Removes the specified annotation from the collection.
◆ void	<u>RemoveAll</u> ()	Removes all annotations from the collection.

## Examples

---

1. Set the text and visibility of the first annotation on the graph.

```
CNiGraph graph;  
CNiAnnotation annotation = graph.Annotations.Item(1);  
annotation.Caption.Text = "Critical Point";  
annotation.Visible = true;
```

2. Set the arrow head style.

```
CNiGraph graph;  
graph.Annotations.Item(1).Arrow.HeadStyle = CNiArrow::Solid;
```

3. Add an additional annotation to the graph and hide it.

```
CNiGraph graph;  
CNiAnnotation annotation = graph.Annotations.Add();  
annotation.Visible = false;
```

4. Remove all annotations from the graph.

```
CNiGraph graph;  
graph.Annotations.RemoveAll();
```

5. Change the name of the first annotation.

```
CNiGraph graph;  
graph.Annotations.Item(1).Name = "Time";
```

Subsequent accesses to this annotation must now use "Time" for its name. For example,

```
graph.Annotations.Item("Time").Visible = true;
```

---

 **CNiArrow**



**Class**

Declared in:  
**NiArrow.h**





## Overview

---

CNiArrow encapsulates the interface to a single arrow on an annotation on a graph control, which allows you to modify the appearance and behavior of the arrow.

You obtain individual arrows using the Arrow property on a CNiAnnotation object.

 [Hierarchy Chart](#)

## ▲ Base Classes

---

### ▲ CNiInterface

## ◆ Data Items

---

- |                   |                  |  |
|-------------------|------------------|--|
| ◆ <u>CNiColor</u> | <u>Color</u>     | Specifies the color of the arrow.            |
| ◆ ArrowHeadStyles | <u>HeadStyle</u> | Specifies the style of the arrow head.       |
| ◆ LineStyles      | <u>LineStyle</u> | Specifies the line style of the arrow.       |
| ◆ ArrowHeadStyles | <u>TailStyle</u> | Specifies the style of the arrow tail.       |
| ◆ <u>bool</u>     | <u>Visible</u>   | Specifies if the arrow is visible or hidden. |
| ◆ <u>short</u>    | <u>Width</u>     | Specifies the width of the arrow.            |

## ◆ Constructors

---

◆ **CNiArrow()**

Default constructor.

◆ **CNiArrow**( CWArrow\_CI\* pCustom,  
[CniInterface::ThreadAccess](#) option )

Constructor that attaches to the  
specified CWArrow\_CI pointer.

◆ **CNiArrow**( [const](#) CNiArrow&  
source )

Copy constructor.

## ◆ **D e s t r u c t o r s**

---

◆ `~CNiArrow()`

Destructor.

## ◆ **F u n c t i o n s**

---

- ◆ `static const IID & GetIid()` Returns the globally unique identifier (GUID) of the ActiveX interface to which this class connects.
- ◆ `const CNiArrow & operator =(  
const  
CNiArrow&  
source )` Assignment operator.

## Example

---

// Change the visibility of the arrow on the first annotation.

```
CNiGraph graph;  
CNiArrow arrow = graph.Annotations.Item(1).Arrow;  
arrow.Visible = false;
```

---

# **CNiAxes**



**Class**

Declared in:  
**NiAxes.h**





## Overview

---

A `CNiAxes` object is a collection of axes on a 2D graph control. This collection always contains one x axis and at least one y axis, although it may contain multiple y axes.

- Use the `CNiGraph::Axes` property to get the axis collection for the graph.
- Use the `Add` function to create additional y axes. `Add` returns a `CNiAxis` object, which represents the new axis.
- Use the `Item` function to access existing axes in the collection. You use this function to access axes by either name or index.
- Use the `Remove` function to remove existing axes from the collection. You use this function to access axes by either name or index. A `COleException` is thrown if you try to remove the x axis or the last y axis.
- Use the `RemoveAll` function to remove all optional y axes from the collection.

 [Hierarchy Chart](#)

## ▲ Base Classes

---

### ▲ CNiInterface

## ◆ Data Items

---

◆ [short](#)     [Count](#)     The number of axes in the collection.

## ◆ Constructors

---

- ◆ [CNIAxes\(\)](#) Default constructor.
- ◆ [CNIAxes](#)( CWAxes\_CI\* pCustom, [CNIInterface::ThreadAccess](#) option ) Constructor that attaches to the specified CWAxes\_CI pointer.
- ◆ [CNIAxes](#)( [const](#) CNIAxes& source ) Copy constructor.

## ◆ **D e s t r u c t o r s**

---

◆ `~CNiAxes()`

Destructor.

## ◆ Functions

---

- ◆ CNiAxis            **Add()**            Adds a new axis to the collection and returns the new axis.
- ◆ `static const IID &` **GetIid()**            Returns the globally unique identifier (GUID) of the ActiveX interface to which this class connects.
- ◆ CNiAxis            **Item**( long axisIndex )            Returns the specified axis from the collection.
- ◆ CNiAxis            **Item**( const CString& axisName )            Returns the specified axis from the collection.
- ◆ `const CNiAxes &` **operator =**( const CNiAxes& source )            Assignment operator.
- ◆ `void`            **Remove**( long axisIndex )            Removes the specified axis from the collection.
- ◆ `void`            **Remove**( const CString& axisName )            Removes the specified axis from the collection.
- ◆ `void`            **RemoveAll**()            Removes all axes from the collection except for the default x and y axes.

## Examples

---

1. Set the caption and visibility of the x axis.

```
CNiGraph graph;  
CNiAxis xAxis = graph.Axes.Item("XAxis");  
xAxis.Caption = "Temperature";  
xAxis.Visible = true;
```

2. Set the range of the x axis on the graph from 0 to 100.

```
CNiGraph graph;  
graph.Axes.Item(1).SetMinMax(0, 100);
```

3. Add an additional y axis to the graph and set its caption.

```
CNiGraph graph;  
CNiAxis axis = graph.Axes.Add();  
axis.Caption = "Power";
```

4. Remove all optional y axes.

```
CNiGraph graph;  
graph.Axes.RemoveAll();
```

5. Change the name of the x axis to "Time".

```
CNiGraph graph;  
graph.Axes.Item("XAxis").Name = "Time";
```

Subsequent attempts to access the X Axis must now use "Time" as the item name. For example,

```
graph.Axes.Item("Time").Visible = true;
```

---

# **CNiAxis**



**Class**

Declared in:  
**NiAxis.h**





## Overview

---

CNiAxis encapsulates the interface to a single axis of a graph, knob, or slide control, which allows you to modify the appearance and behavior of the axis.

You obtain individual axes using the Axes property on a CNiGraph object.

Note: To specify a date/time value, you must convert your date or time value to a double. A date is implemented as a floating-point value with the integer part of the number measuring days from midnight, 30 December 1899, and the fractional part representing the time of day. The absolute value of the fractional part of the number represents the time as a fraction of a day. Thus, 1 second equals  $1 / 24 \text{ hours} / 60 \text{ minutes}$ , which is  $1/86400$  or approximately  $1.157407\text{e-}5$ . So, midnight, 31 December 1899, is represented by 1.0. Similarly, 6 AM, 1 January 1900, is represented by 2.25, and midnight, 29 December 1899, is -1.0. However, 6 AM, 29 December 1899, is -1.25.

 [Hierarchy Chart](#)

## ▲ Base Classes

---

### ▲ CNiInterface

## ◆ Data Items

---

◆ <a href="#">bool</a>	<a href="#"><u>AutoScale</u></a>	Determines if the system automatically sets the minimum and maximum limits of the axis.
◆ <a href="#">CString</a>	<a href="#"><u>Caption</u></a>	Specifies the text to draw on the axis.
◆ <a href="#">CColor</a>	<a href="#"><u>CaptionColor</u></a>	Specifies the color used to draw the caption.
◆ <a href="#">bool</a>	<a href="#"><u>Discrete</u></a>	Represents only discrete values on the axis, according to the base and interval properties.
◆ <a href="#">double</a>	<a href="#"><u>DiscreteBase</u></a>	Specifies the base value for discrete axes.
◆ <a href="#">double</a>	<a href="#"><u>DiscreteInterval</u></a>	Specifies the interval between discrete values.
◆ <a href="#">CString</a>	<a href="#"><u>FormatString</u></a>	Specifies the format string for formatting the labels on this axis.
◆ <a href="#">bool</a>	<a href="#"><u>Inverted</u></a>	Specifies if the direction of an axis is inverted.
◆ <a href="#">CLabels</a>	<a href="#"><u>Labels</u></a>	Returns a <code>CLabel</code> object, which specifies how labels appear on the axis.
◆ <a href="#">bool</a>	<a href="#"><u>Log</u></a>	Specifies if the axis has a Log10 scale.
◆ <a href="#">double</a>	<a href="#"><u>Maximum</u></a>	Specifies the maximum value of the axis.
◆ <a href="#">double</a>	<a href="#"><u>Minimum</u></a>	Specifies the minimum value of the axis.
◆ <a href="#">CString</a>	<a href="#"><u>Name</u></a>	Specifies the name of the axis.
◆ <a href="#">CTicks</a>	<a href="#"><u>Ticks</u></a>	Returns a <code>CTicks</code> object, which specifies how divisions and ticks appear on this axis.
◆ <a href="#">CValuePairs</a>	<a href="#"><u>ValuePairs</u></a>	Gets a collection of <code>ValuePair</code> objects associated with the control.
◆ <a href="#">bool</a>	<a href="#"><u>Visible</u></a>	Specifies if the axis is visible or hidden.

## ◆ Constructors

---

- ◆ [CNiAxis\(\)](#) Default constructor.
- ◆ [CNiAxis](#)( CWAxis\_CI\* pCustom, [CNiInterface::ThreadAccess](#) option ) Constructor that attaches to the specified CWAxis\_CI pointer.
- ◆ [CNiAxis](#)( [const](#) CNiAxis& source ) Copy constructor.

## ◆ **D e s t r u c t o r s**

---

◆ `~CNiAxis()`

Destructor.

## ◆ Functions

---

- ◆ void **AutoScaleNow()** Causes the axis to rescale immediately.
- ◆ static const IID & **GetIid()** Returns the globally unique identifier (GUID) of the ActiveX interface to which this class connects.
- ◆ const CNiAxis & operator =( const CNiAxis& source ) Assignment operator.
- ◆ void **SetMinMax( double Minimum, double Maximum )** Sets both the minimum and the maximum values of the axis at the same time.

## Example

---

```
// Set the caption and visibility of the x axis.
```

```
CNiGraph graph;  
CNiAxis xAxis = graph.Axes.Item("XAxis");  
xAxis.Caption = "Temperature";  
xAxis.Visible = true;
```

---

# **CNiBinding**



**Class**

Declared in:  
**NiBinding.h**





## Overview

---

CNiBinding encapsulates the interface to a single binding object of a button, graph, knob, numeric edit, or slide control. This allows you bind a control property to a data source, such as an item on an OPC or DataSocket Server.

You can use this object to automatically associate a property with a data item. For example, if you want a slide to always display the value of the data item "wave" from a DataSocket Server, you set up a binding to connect the CNiSlide::Value property to "dstp://localhost/wave".

Notes:

1. You can bind any control property, not just the value. For example, you can bind the graph background color to an item on an OPC Server.
2. You also can manipulate the data values in the CWBindingOnDataUpdated event on each control, so that you can scale the values being sent or received from the property values.
3. For a complete list of a control's bindable properties, refer to the Bindings property page for that control.
4. You must call CNiBinding::SetBindObject in the same thread that you used to create the object to which you are binding.
5. Currently, data binding is not supported on the 3D graph control.
6. You must initialize a CNiBinding object from an existing object. If you do not initialize a CNiBinding object from an existing object, a CNiObjectNotInUsableState exception will be thrown when you attempt to manipulate the instance of the CNiBinding.

 [Hierarchy Chart](#)

## ▲ Base Classes

---

### ▲ CNiInterface

## ◆ Data Items

---

◆ AccessModes	<u>AccessMode</u>	Specifies the access mode for the connection.
◆ CString	<u>ActualURL</u>	Identifies the actual URL of the current data source or data target.
◆ CString	<u>BindProperty</u>	The name of the property to bind to a data source or data target.
◆ bool	<u>DataUpdated</u>	Specifies if the CNIBinding object's data value or attributes have been set since they were last read or written.
◆ bool	<u>DataUpdatedEnabled</u>	Specifies if the binding should generate the DataUpdated event whenever the binding data changes.
◆ long	<u>LastError</u>	The last error code used in the StatusUpdated event.
◆ CString	<u>LastMessage</u>	The last message used in the StatusUpdated event.
◆ ConnectionStatus	<u>Status</u>	The current status of the CNIBinding connection.
◆ bool	<u>StatusUpdated</u>	The CNIBinding object calls this function when its Status property is updated.
◆ long	<u>TimerInterval</u>	Use this property to configure CNIBinding to automatically call update at regular intervals.
◆ CString	<u>Url</u>	Specifies the data source or target.

## ◆ Constructors

---

- ◆ **CNiBinding**() Default constructor.
- ◆ **CNiBinding**( CWBinding\_CI\* pCustom, CNiInterface::ThreadAccess option ) Constructor that attaches to the specified CWBinding\_CI pointer.
- ◆ **CNiBinding**( const CNiBinding& source ) Copy constructor.

## ◆ **D e s t r u c t o r s**

---

◆ `~CNiBinding()`

Destructor.

## ◆ Functions

---

- |                      |   |  |
|----------------------|---|--|
| ◆ void               | <b>Connect</b> ( LPCTSTR URL, AccessModes accessMode )  | Connects the CNIBinding object to a data source or target.   |
| ◆ void               | <b>Connect</b> ()   | Connects the CNIBinding object to a data source or target.   |
| ◆ void               | <b>Disconnect</b> ()  | Disconnect the binding connection from the source to which it is currently connected.                          |
| ◆ static const IID & | <b>GetId</b> ()   | Returns the globally unique identifier (GUID) of the ActiveX interface to which this class connects.           |
| ◆ const CNIBinding & | <b>operator =</b> ( const CNIBinding& source )  | Assignment operator.   |
| ◆ bool               | <b>SelectURL</b> ( const CString& startURL = "", const CString& title = "", int options = 0, const CString& filter = "" ) | Displays a dialog box for the user to select a data source/target and sets the URL property to that data item. |
| ◆ void               | <b>SetBindObject</b> ( CNIInterface& subObject )  | Sets the object whose property binds to a data source.   |
| ◆ void               | <b>SetBindObject</b> ( CNIControl& control )  | Sets the object whose property binds to a data source.   |
| ◆ void               | <b>Update</b> ()  | Causes the CNIBinding object to read from a data source or write to a data target.                             |
-

# C Ni Bindings



Class

Declared in:  
**NiBindings.h**





## Overview

---

A `CNiBindings` object is a collection of data binding objects on a button, graph, knob, numeric edit, or slide control. Data binding allows you bind a control property to a data source, such as an item on an OPC or DataSocket Server.

- Use the `Bindings` property on the control to obtain this collection.
- Use the `Add` function to create additional binding. `Add` returns a `CNiBinding` object, which represents the new data binding.
- Use the `Item` function to access existing bindings in the collection. This function can access bindings by either name or index.
- Use the `Remove` function to remove existing bindings from the collection. This function can access bindings by either name or index.
- Use the `RemoveAll` function to remove all bindings from the collection.

 [Hierarchy Chart](#)

## ▲ Base Classes

---

### ▲ CNiInterface

## ◆ Data Items

---

◆ **short**      Count    The number of objects in the collection.

## ◆ Constructors

---

- ◆ **CNiBindings()** Default constructor.
- ◆ **CNiBindings**( CWBindings\_CI\* pCustom, [CNiInterface::ThreadAccess](#) option ) Constructor that attaches to the specified CWBindings\_CI pointer.
- ◆ **CNiBindings**( [const](#) CNiBindings& source ) Copy constructor.

## ◆ **D e s t r u c t o r s**

---

◆ `~CNIBindings()`

Destructor.



# C NiButton



**Class**

Declared in:  
**NiButton.h**





## Overview

---

CNiButton encapsulates the interface to the Measurement Studio ActiveX button control, which represents different Boolean displays such as on/off or true/false. Typically, you often use buttons to input or output Boolean information or initiate an action in your program.

CNiButton is capable of responding to events that are generated by the control. For a list of the events that can be generated by this control and details on how to respond to the events in your program, refer to the [Button Events](#) page.

### Features

- Different display styles - toggle switches, LEDs, push buttons, slides, and on/off buttons.
- Custom bitmap buttons.
- Button modes specify how the Button control responds to user input. For example, you can make a button respond only programmatically (you cannot make a button respond to user input). Or you can click the button to temporarily change its value and then release to revert the button to its original state (called a latch). Finally, you can click on the button to change its value until you click on it again.
- Built-in format styles for the labels, including scientific, symbolic, engineering scaling, time, and date.
- Animation - you can animate different parts of the control.
- Custom background images.
- CNiButton includes bindable properties. You can bind these properties to a DataSocket source or target. This allows you to read property values from and write property values to the source or target. Click [here](#) for a list of bindable properties.

 [Hierarchy Chart](#)

## ▲ Base Classes

---

### ▲ CNiControl

## ◆ Data Items

---

◆ <a href="#">CNiColor</a>	<b><a href="#">BackColor</a></b>	Specifies the background color of the button.
◆ <a href="#">CNIBindings</a>	<b><a href="#">Bindings</a></b>	Gets a collection of binding objects associated with the control.
◆ <a href="#">CString</a>	<b><a href="#">Caption</a></b>	Specifies the text that appears in the button.
◆ <a href="#">CNiColor</a>	<b><a href="#">CaptionColor</a></b>	Specifies the color of the caption for the button.
◆ <a href="#">bool</a>	<b><a href="#">Enabled</a></b>	Specifies if the control responds to user input.
◆ <a href="#">CNIFont</a>	<b><a href="#">Font</a></b>	Specifies the font for the caption and the two text fields in the button.
◆ <a href="#">bool</a>	<b><a href="#">ImmediateUpdates</a></b>	Specifies if the control draws new data as soon as it is available or if the form refreshes the control when it draws other controls.
◆ KeyboardModes	<b><a href="#">KeyboardMode</a></b>	Specifies how the control handles keyboard input from the user.
◆ ButtonModes	<b><a href="#">Mode</a></b>	Specifies how the button responds to user input.
◆ <a href="#">CNiColor</a>	<b><a href="#">OffColor</a></b>	Specifies the color of the button in the off state.
◆ <a href="#">CString</a>	<b><a href="#">OffText</a></b>	Specifies the text for the button in the off state.
◆ <a href="#">CNiColor</a>	<b><a href="#">OffTextColor</a></b>	Specifies the color of the text for the button in the off state.
◆ <a href="#">CNiColor</a>	<b><a href="#">OnColor</a></b>	Specifies the color of the button in the on state.
◆ <a href="#">CString</a>	<b><a href="#">OnText</a></b>	Specifies the text for the button in the on state.



<u>CNiColor</u>	<b><u>OnTextColor</u></b>	Specifies the color of the text for the button in the on state.
◆ long	<b><u>ReadyState</u></b>	Returns the ready state of the control.
◆ ShowFocusModes	<b><u>ShowFocusMode</u></b>	Specifies how the control indicates it has the focus.
◆ bool	<b><u>Value</u></b>	Specifies the current value of the button.
◆ bool	<b><u>Windowless</u></b>	Specifies if the control has a window.

## ◆ Constructors

---

◆ **CNiButton**( [CNiInterface::ThreadAccess](#) option = [CNiInterface::MultipleThreadsWithCaching](#) )

## ◆ **D e s t r u c t o r s**

---

◆ ~CNiButton()

## ◆ Functions

---

◆ void	<a href="#"><u>AboutBox()</u></a>	Displays the About Box for the control.
◆ CNIPicture	<a href="#"><u>ControllImage()</u></a>	Returns an image of the entire control.
◆ virtual BOOL	<a href="#"><u>Create</u></a> ( LPCTSTR lpszClassName, LPCTSTR lpszWindowName, DWORD dwStyle, <b>const</b> RECT& rect, <a href="#"><u>CWnd*</u></a> pParentWnd, UINT nID, CCreateContext* pContext = NULL )	Creates the ActiveX control that is represented in the MFC program by this object.
◆ BOOL	<a href="#"><u>Create</u></a> ( LPCTSTR lpszWindowName, DWORD dwStyle, <b>const</b> RECT& rect, <a href="#"><u>CWnd*</u></a> pParentWnd, UINT nID, CFile* pPersist = NULL, BOOL bStorage = FALSE, BSTR bstrLicKey = NULL )	Creates the ActiveX control that is represented in the MFC program by this object.
◆ BOOL	<a href="#"><u>CreateControl</u></a> ( LPCTSTR lpszClass, LPCTSTR lpszWindowName, DWORD dwStyle, <b>const</b> RECT& rect, <a href="#"><u>CWnd*</u></a> pParentWnd, UINT nID, CFile* pPersist = NULL, BOOL bStorage = FALSE, BSTR bstrLicKey = NULL )	Creates the ActiveX control that is represented in the MFC program by this object.
◆ BOOL	<a href="#"><u>CreateControl</u></a> ( REFCLSID clsid, LPCTSTR lpszWindowName, DWORD dwStyle, <b>const</b> RECT& rect, <a href="#"><u>CWnd*</u></a> pParentWnd, UINT nID, CFile* pPersist = NULL, BOOL bStorage = FALSE, BSTR bstrLicKey = NULL )	Creates the ActiveX control that is represented in the MFC program by this object.
◆ void	<a href="#"><u>ExportStyle</u></a> ( LPCTSTR fileName )	Exports the style of the control to a file.



`static const CLSID & GetClsid()`

Returns the globally unique identifier (GUID) of the ActiveX control to which this class connects.

◆ `static const IID & GetIid()`

Returns the globally unique identifier (GUID) of the ActiveX interface to which this class connects.

◆ `void ImportStyle( LPCTSTR fileName )`

Imports a previously exported style.

◆ `CNiImage OffImages( long imageIndex )`

Gets the image object corresponding to the Off state of the `CNiButton` control that is associated with the specified part of the `CNiButton` control.

◆ `CNiImage OffImages( const CString& imageName )`

Gets the image object corresponding to the Off state of the `CNiButton` control that is associated with the specified part of the `CNiButton` control.

◆ `CNiImage OnImages( long imageIndex )`

Gets the image object corresponding to the On state of the `CNiButton` control that is associated with the specified part of the `CNiButton` control.

◆ CNiImage

OnImages( const CString&  
imageName )

Gets the image object corresponding to the On state of the CNiButton control that is associated with the specified part of the CNiButton control.

◆ void

Refresh()

Forces the control to redraw.

◆ void

SetBuiltinStyle( ButtonStyles  
style )

Sets many properties of the control to represent the new style specified.

◆ void

ValidateControl()

Validates the current state of the control.

---

# **CNiCaption**



**Class**

Declared in:  
**NiCaption.h**





## Overview

---

CNiCaption encapsulates the interface to an annotation caption on a graph control, which allows you to modify its appearance and behavior.

You obtain a caption using the `Caption` property on a `CNiAnnotation` object.

Note: To specify a date/time value, you must convert your date or time value to a double. A date is implemented as a floating-point value with the integer part of the number measuring days from midnight, 30 December 1899, and the fractional part representing the time of day. The absolute value of the fractional part of the number represents the time as a fraction of a day. Thus, 1 second equals  $1 / 24 \text{ hours} / 60 \text{ minutes}$ , which is  $1/86400$  or approximately  $1.157407\text{e-}5$ . So, midnight, 31 December 1899, is represented by 1.0. Similarly, 6 AM, 1 January 1900, is represented by 2.25, and midnight, 29 December 1899, is -1.0. However, 6 AM, 29 December 1899, is -1.25.

 [Hierarchy Chart](#)

## ▲ Base Classes

---

### ▲ CNiInterface

## ◆ Data Items

---

- ◆ Alignments **Alignment** Specifies the caption text alignment relative to CNiCaption::XCoordinate and CNiCaption::YCoordinate.
- ◆ double **Angle** Angle of the caption text in degrees.
- ◆ CNiColor **Color** Specifies the color of the caption.
- ◆ CNiFont **Font** Specifies the font of the caption.
- ◆ CString **Text** Specifies the the caption text.
- ◆ double **XCoordinate** X coordinate of the caption.
- ◆ double **YCoordinate** Y coordinate of the caption.

## ◆ Constructors

---

- ◆ **CNiCaption()** Default constructor.
- ◆ **CNiCaption**( CWCaption\_CI\* pCustom, **CNiInterface::ThreadAccess** option ) Constructor that attaches to the specified CWCaption\_CI pointer.
- ◆ **CNiCaption**( **const** CNiCaption& source ) Copy constructor.

## ◆ **D e s t r u c t o r s**

---

◆ `~CNiCaption()`

Destructor.

## ◆ **F u n c t i o n s**

---

- ◆ `static const IID & GetIid()` Returns the globally unique identifier (GUID) of the ActiveX interface to which this class connects.
- ◆ `const CNiCaption & operator =( const CNiCaption& source )` Assignment operator.
- ◆ `void SetCoordinates( double xCoordinate, double yCoordinate )` Sets the x and y coordinates of the caption at the same time.

## Example

---

```
// Change the text of the first annotation on the graph.
```

```
CniGraph graph;  
CniCaption caption = graph.Annotations.Item(1).Caption;  
caption.Text = "Maximum";
```

---

# **CNiControlMetrics**



**Class**

Declared in:  
**NiControlMetrics.h**





## Overview

---

CNiControlMetrics encapsulates the control metrics property of a knob or slider control.

These metrics include the following elements.

- Minimum pointer value.
- Maximum pointer value.
- Mean pointer value of last ten pointer values.

Use the Reset function to reset the metrics.

 [Hierarchy Chart](#)

## ▲ Base Classes

---

### ▲ CNiInterface

## ◆ Data Items

---

- ◆ double     **Maximum** Returns the maximum value of any pointer since the metrics were last reset.
- ◆ double     **Mean** Returns the mean of the last 10 pointer values.
- ◆ double     **Minimum** Returns the minimum value of any pointer since the metrics were last reset.

## ◆ Constructors

---

- ◆ [CNiControlMetrics\(\)](#) Default constructor.
- ◆ [CNiControlMetrics](#)( CWStatistics\_CI\* pCustom, [CNiInterface::ThreadAccess](#) option ) Constructor that attaches to the specified CWStatistics\_CI pointer.
- ◆ [CNiControlMetrics](#)( [const](#) CNiControlMetrics& source ) Copy constructor.

## ◆ **D e s t r u c t o r s**

---

◆ `~CNiControlMetrics()`

Destructor.

## ◆ **F u n c t i o n s**

---

- ◆ `static const IID &` **GetIid()** Returns the globally unique identifier (GUID) of the ActiveX interface to which this class connects.
  - ◆ `const CNiControlMetrics & operator =( const CNiControlMetrics& source )` Assignment operator.
  - ◆ `void` **Reset()** Resets the minimum, maximum, and mean metrics.
-



**Class**

Declared in:  
**NiCursor.h**





## Overview

---

CNiCursor encapsulates the interface to a single cursor on a graph control, which allows you to modify its appearance and behavior.

Individual cursors are obtained via the `Cursors` property on a `CNiGraph` object.

Note: To specify a date/time value, you must convert your date or time value to a double. A date is implemented as a floating-point value with the integer part of the number measuring days from midnight, 30 December 1899, and the fractional part representing the time of day. The absolute value of the fractional part of the number represents the time as a fraction of a day. Thus, 1 second equals  $1 / 24 \text{ hours} / 60 \text{ minutes}$ , which is  $1/86400$  or approximately  $1.157407\text{e-}5$ . So, midnight, 31 December 1899, is represented by 1.0. Similarly, 6 AM, 1 January 1900, is represented by 2.25, and midnight, 29 December 1899, is -1.0. However, 6 AM, 29 December 1899, is -1.25.

 [Hierarchy Chart](#)

## ▲ Base Classes

---

### ▲ CNiInterface

## ◆ Data Items

---

◆ <a href="#">CNiColor</a>	<b><u>Color</u></b>	Specifies the color of the cursor crosshair and point.
◆ CrosshairStyles	<b><u>CrosshairStyle</u></b>	Specifies the type of lines that identify the cursor position.
◆ <a href="#">bool</a>	<b><u>Enabled</u></b>	Specifies if the cursor generates mouse events or if you can drag the cursor in cursor tracking mode.
◆ <a href="#">CString</a>	<b><u>Name</u></b>	Specifies the name of the cursor.
◆ <a href="#">CniPlot</a>	<b><u>Plot</u></b>	Specifies the plot associated with the cursor.
◆ <a href="#">long</a>	<b><u>PointIndex</u></b>	Specifies the point associated with the cursor on the plot.
◆ PointStyles	<b><u>PointStyle</u></b>	Specifies the cursor point style.
◆ SnapModes	<b><u>SnapMode</u></b>	Specifies the snap mode of the cursor.
◆ <a href="#">bool</a>	<b><u>Visible</u></b>	Specifies if the cursor is visible or hidden.
◆ <a href="#">double</a>	<b><u>XPosition</u></b>	Current x axis position of the cursor.
◆ <a href="#">double</a>	<b><u>YPosition</u></b>	Current y axis position of the cursor.

## ◆ Constructors

---

- ◆ **CNiCursor()** Default constructor.
- ◆ **CNiCursor**( CWCursor\_CI\* pCustom, **CNiInterface::ThreadAccess** option ) Constructor that attaches to the specified CWCursor\_CI pointer.
- ◆ **CNiCursor**( **const** CNiCursor& source ) Copy constructor.

## ◆ **D e s t r u c t o r s**

---

◆ `~CNiCursor()`

Destructor.

## ◆ **F u n c t i o n s**

---

- ◆ `static const IID & GetIid()` Returns the globally unique identifier (GUID) of the ActiveX interface to which this class connects.
- ◆ `const CNiCursor & operator =( const CNiCursor& source )` Assignment operator.
- ◆ `void SetPosition( double xPosition, double yPosition )` Sets the x and y axis positions of the cursor at the same time.

## Example

---

```
// Change the snap mode of the first cursor on the graph.
```

```
CNiGraph graph;
```

```
CNiCursor cursor = graph.Cursors.Item(1);
```

```
cursor.SnapMode = CNiCursor::SnapNearestPoint;
```

---



**Class**

Declared in:  
**NiCursors.h**





## Overview

---

CNiCursors encapsulates the interface to the Cursors property of a CNiGraph object, which allows you to access and remove cursors associated with the graph control.

 [Hierarchy Chart](#)

## ▲ Base Classes

---

### ▲ CNiInterface

## ◆ Data Items

---

◆ [short](#)      Count Returns the number of cursors in the collection.

## ◆ **C o n s t r u c t o r s**

---

- ◆ **CNiCursors()** Default constructor.
- ◆ **CNiCursors**( CWCursors\_CI\* pCustom, **CNiInterface::ThreadAccess** option ) Constructor that attaches to the specified cWCursors\_CI pointer.
- ◆ **CNiCursors**( **const** CNiCursors& source ) Copy constructor.

## ◆ **D e s t r u c t o r s**

---

◆ `~CNCursors()`

Destructor.



## ▣ Examples

---

1. Set the snap mode and visibility of the first cursor on the graph.

```
CNiGraph graph;  
CNiCursor cursor = graph.Cursors.Item(1);  
cursor.SnapMode = CNiCursor::SnapNearestPoint;  
cursor.Visible = true;
```

2. Set the crosshair style.

```
CNiGraph graph;  
graph.Cursors.Item(1).CrosshairStyle = CNiCursor::CrosshairMinorXMinor
```

3. Add an additional cursor to the graph and hide it.

```
CNiGraph graph;  
CNiCursor cursor = graph.Cursors.Add();  
cursor.Visible = false;
```

4. Remove all cursors from the graph.

```
CNiGraph graph;  
graph.Cursors.RemoveAll();
```

5. Change the name of the first cursor.

```
CNiGraph graph;  
graph.Cursors.Item(1).Name = "Time";
```

Subsequent accesses to this cursor must now use "Time" for its name. For example,

```
graph.Cursors.Item("Time").Visible = true;
```

---

# **CNiGraph**



**Class**

Declared in:  
**NiGraph.h**





## Overview

---

CNiGraph encapsulates the interface to the Measurement Studio ActiveX graph control, which allows you to plot and chart two-dimensional data.

CNiGraph is capable of responding to events that are generated by the control. For a list of the events that can be generated by this control and details on how to respond to the events in your program, refer to the [Graph Events](#) page.

### Features

- Plotting and Charting - plotting data refers to the process of taking a large number of points and updating one or more plots on the graph with new data, replacing the old plot with the new plot. Charting data appends new data points to an existing plot over time. Charting is used with slow processes where only few data points per second are added to the graph. When the number of data points exceeds the number of points that can be displayed on the graph, the graph scrolls so that new points are added to the right side of the graph while old points disappear to the left.
- Multiple plot styles - point, line, line-point, and bar.
- Multiple plots with individual properties such as name, line and point style, width, and base value.
- Cursors - display a crosshair on a graph to mark a specific point or region on the graph or highlight data.
- Configurable axes, including customizable ticks, labels, value pairs, and captions.
- Built-in format styles for labels, including scientific, symbolic engineering, scaling, time, and date.
- Panning and zooming at runtime - panning is useful when the graph displays only a subset of the data that has been plotted. You can scroll through all data plotted on the graph, essentially shifting the graph's display to different portions of the plot.

Note: Set the `TrackMode` property to allow user interaction (such as panning, zooming, moving cursors, and moving annotations) with the graph while your program is running.

- CNiGraph includes bindable properties. You can bind these properties to a `DataSocket` source or target. This allows you to read property values from and write property values to the source or target. Click [here](#) for a

list of bindable properties.

Note: To specify a date/time value, you must convert your date or time value to a double. A date is implemented as a floating-point value with the integer part of the number measuring days from midnight, 30 December 1899, and the fractional part representing the time of day. The absolute value of the fractional part of the number represents the time as a fraction of a day. Thus, 1 second equals  $1 / 24 \text{ hours} / 60 \text{ minutes}$ , which is  $1/86400$  or approximately  $1.157407\text{e-}5$ . So, midnight, 31 December 1899, is represented by 1.0. Similarly, 6 AM, 1 January 1900, is represented by 2.25, and midnight, 29 December 1899, is -1.0. However, 6 AM, 29 December 1899, is -1.25.

 [Hierarchy Chart](#)

## ▲ Base Classes

---

### ▲ CNiControl

## ◆ Data Items

---

◆ <a href="#">CNIAnnotations</a>	<a href="#"><u>Annotations</u></a>	Gets a collection of annotation objects associated with the control.
◆ <a href="#">CNIAnnotation</a>	<a href="#"><u>AnnotationTemplate</u></a>	Returns the annotation to use as a template for new annotations.
◆ <a href="#">CNIAxes</a>	<a href="#"><u>Axes</u></a>	Gets a collection of axis objects associated with the control.
◆ <a href="#">CNIColor</a>	<a href="#"><u>BackColor</u></a>	Specifies the color for the graph caption's background.
◆ <a href="#">CNIBindings</a>	<a href="#"><u>Bindings</u></a>	Gets a collection of binding objects associated with the control.
◆ <a href="#">CString</a>	<a href="#"><u>Caption</u></a>	Specifies the caption to be drawn on the CNIgraph.
◆ <a href="#">CNIColor</a>	<a href="#"><u>CaptionColor</u></a>	Specifies the color of the caption.
◆ <a href="#">long</a>	<a href="#"><u>ChartLength</u></a>	Specifies how many points the graph stores when charting before deleting old data.
◆ <a href="#">ChartStyles</a>	<a href="#"><u>ChartStyle</u></a>	Specifies how chart functions update the display as new data is added to the plot.
◆ <a href="#">CNI Cursors</a>	<a href="#"><u>Cursors</u></a>	Gets a collection of cursor objects associated with the control.
◆ <a href="#">bool</a>	<a href="#"><u>DefaultPlotPerRow</u></a>	This is the default value used by the overloaded versions of the Plot/Chart functions that do not take the plotPerRow parameter.
◆ <a href="#">double</a>	<a href="#"><u>DefaultXFirst</u></a>	This is the default value used by the overloaded versions of the PlotY functions that do not

◆ double	<b><u>DefaultXInc</u></b>	take the xFirst parameter. This is the default value used by the overloaded versions of the PlotY and ChartY functions that do not take the xInc parameter.
◆ bool	<b><u>Enabled</u></b>	Specifies if the graph generates any events.
◆ <u>CNiFont</u>	<b><u>Font</u></b>	Specifies the font for labels on all axes.
◆ <u>CNiColor</u>	<b><u>GraphFrameColor</u></b>	Specifies the color for the graph frame.
◆ GraphFrameStyles	<b><u>GraphFrameStyle</u></b>	Determines if the graph has a 3D frame or a classic frame.
◆ bool	<b><u>ImmediateUpdates</u></b>	Specifies if the graph draws new data as soon as it is available.
◆ KeyboardModes	<b><u>KeyboardMode</u></b>	Specifies how the control handles keyboard input from the user.
◆ <u>CNiColor</u>	<b><u>PlotAreaColor</u></b>	Specifies the background color of the plot area.
◆ <u>CNiPlots</u>	<b><u>Plots</u></b>	Gets a collection of plot objects associated with the control.
◆ <u>CNiPlot</u>	<b><u>PlotTemplate</u></b>	Returns the plot to use as a template for new plots.
◆ long	<b><u>ReadyState</u></b>	Returns the ready state of the control.
◆ GraphTrackModes	<b><u>TrackMode</u></b>	Determines how the mouse interacts with the graph.
◆ bool	<b><u>Windowless</u></b>	Specifies if the control has a window.

## ◆ Constructors

---

◆ **CNiGraph**( [cniInterface::ThreadAccess](#) option = [cniInterface::MultipleThreadsWithCaching](#) )

## ◆ **D e s t r u c t o r s**

---

◆ `~CNiGraph()`

## ◆ Functions

---

◆ void	<a href="#"><u>AboutBox()</u></a>	Displays the About Box for the control.
◆ void	<a href="#"><u>ChartXvsY(const CNiVector&amp; x, const CNiMatrix&amp; y, bool chartPerRow )</u></a>	Charts a matrix of data as one or more x-y plots.
◆ void	<a href="#"><u>ChartXvsY(const CNiVector&amp; x, const CNiMatrix&amp; y )</u></a>	Charts a matrix of data as one or more x-y plots.
◆ void	<a href="#"><u>ChartXvsY(const CNiVector&amp; x, const CNiVector&amp; y )</u></a>	Charts two vectors of data as a single x-y plot.
◆ void	<a href="#"><u>ChartXY(const CNiMatrix&amp; xy, bool chartPerRow )</u></a>	Charts a matrix of data as one or more x-y plots.
◆ void	<a href="#"><u>ChartXY(const CNiMatrix&amp; xy )</u></a>	Charts a matrix of data as one or more x-y plots.
◆ void	<a href="#"><u>ChartXY(double x, double y )</u></a>	Charts a single point of data as one x-y plot.
◆ void	<a href="#"><u>ChartY(const CNiMatrix&amp; y, double xInc, bool chartPerRow )</u></a>	Charts a matrix of data as one or more y plots.
◆ void	<a href="#"><u>ChartY(const CNiMatrix&amp; y, bool chartPerRow )</u></a>	Charts a matrix of data as one or more y plots.
◆ void	<a href="#"><u>ChartY(const CNiMatrix&amp; y )</u></a>	Charts a matrix of data as one or more y plots.
◆ void	<a href="#"><u>ChartY(const CNiVector&amp; y, double xInc )</u></a>	Charts a vector of data.
◆ void	<a href="#"><u>ChartY(const CNiVector&amp; y )</u></a>	Charts a vector

◆ void	<u><b>ChartY</b></u> ( double y, double xInc )	of data. Charts a point of data.
◆ void	<u><b>ChartY</b></u> ( double y )	Charts a point of data.
◆ void	<u><b>ClearData</b></u> ()	Clears data in all plots.
◆ CNIPicture	<u><b>ControlImage</b></u> ()	Returns an image of the entire control.
🔑 virtual BOOL	<u><b>Create</b></u> ( LPCTSTR lpszClassName, LPCTSTR lpszWindowName, DWORD dwStyle, <b>const</b> RECT& rect, <u><b>CWnd</b></u> * pParentWnd, UINT nID, CCreateContext* pContext = NULL )	Creates the ActiveX control that is represented in the MFC program by this object.
🔑 BOOL	<u><b>Create</b></u> ( LPCTSTR lpszWindowName, DWORD dwStyle, <b>const</b> RECT& rect, <u><b>CWnd</b></u> * pParentWnd, UINT nID, CFile* pPersist = NULL, BOOL bStorage = FALSE, BSTR bstrLicKey = NULL )	Creates the ActiveX control that is represented in the MFC program by this object.
🔑 BOOL	<u><b>CreateControl</b></u> ( LPCTSTR lpszClass, LPCTSTR lpszWindowName, DWORD dwStyle, <b>const</b> RECT& rect, <u><b>CWnd</b></u> * pParentWnd, UINT nID, CFile* pPersist = NULL, BOOL bStorage = FALSE, BSTR bstrLicKey = NULL )	Creates the ActiveX control that is represented in the MFC program by this object.
🔑 BOOL	<u><b>CreateControl</b></u> ( REFCLSID clsid, LPCTSTR lpszWindowName, DWORD dwStyle, <b>const</b> RECT& rect, <u><b>CWnd</b></u> * pParentWnd, UINT nID, CFile* pPersist = NULL, BOOL bStorage = FALSE, BSTR	Creates the ActiveX control that is represented in the MFC program by this

	bstrLicKey = NULL )	object.
◆ void	<b><u>ExportStyle</u></b> ( LPCTSTR fileName )	Exports the style of the control to a file.
◆ static const CLSID &	<b><u>GetClsid</u></b> ()	Returns the globally unique identifier (GUID) of the ActiveX control to which this class connects.
◆ static const IID &	<b><u>GetIid</u></b> ()	Returns the globally unique identifier (GUID) of the ActiveX interface to which this class connects.
◆ <b><u>CNiImage</u></b>	<b><u>Images</u></b> ( const <b><u>CString</u></b> & imageName )	Gets the image object associated with the specified part of the CNiGraph control.
◆ <b><u>CNiImage</u></b>	<b><u>Images</u></b> ( long imageIndex )	Gets the image object associated with the specified part of the CNiGraph control.
◆ void	<b><u>ImportStyle</u></b> ( LPCTSTR fileName )	Imports a previously exported style.
◆ void	<b><u>PlotXvsY</u></b> ( const <b><u>CNiVector</u></b> & x,	Plots a matrix of

	<code>const <u>CNiMatrix</u>&amp; y, bool plotPerRow )</code>	data as one or more x-y plots.
◆ void	<code><u>PlotXvsY</u>(const <u>CNiVector</u>&amp; x, const <u>CNiMatrix</u>&amp; y )</code>	Plots a matrix of data as one or more x-y plots.
◆ void	<code><u>PlotXvsY</u>(const <u>CNiVector</u>&amp; x, const <u>CNiVector</u>&amp; y )</code>	Plots two vectors of data as a single x-y plot.
◆ void	<code><u>PlotXY</u>(const <u>CNiMatrix</u>&amp; xy, bool plotPerRow )</code>	Plots a matrix of data as one or more x-y plots.
◆ void	<code><u>PlotXY</u>(const <u>CNiMatrix</u>&amp; xy )</code>	Plots a matrix of data as one or more x-y plots.
◆ void	<code><u>PlotXY</u>(double x, double y )</code>	Plots a single point of data as one x-y plot.
◆ void	<code><u>PlotY</u>(const <u>CNiMatrix</u>&amp; y, double xFirst, double xInc, bool plotPerRow )</code>	Plots a matrix of data as one or more y plots.
◆ void	<code><u>PlotY</u>(const <u>CNiMatrix</u>&amp; y, bool plotPerRow )</code>	Plots a matrix of data as one or more y plots.
◆ void	<code><u>PlotY</u>(const <u>CNiMatrix</u>&amp; y )</code>	Plots a matrix of data as one or more y plots.
◆ void	<code><u>PlotY</u>(const <u>CNiVector</u>&amp; y, double xFirst, double xInc )</code>	Plots a vector of data as a single y plot.
◆ void	<code><u>PlotY</u>(const <u>CNiVector</u>&amp; y, double xFirst )</code>	Plots a vector of data as a single y plot.
◆ void	<code><u>PlotY</u>(const <u>CNiVector</u>&amp; y )</code>	Plots a vector of data as a single

◆ void

**Refresh()**

y plot.

Forces the control to redraw.

◆ void

**ValidateControl()**

Validates the current state of the control.

---

# **CNiImage**



**Class**

Declared in:  
**NiImage.h**





## Overview

---

CNiImage encapsulates the interface to an image object that is associated with a Measurement Studio ActiveX control such as a graph or a button. This allows you to manipulate and animate images to create complex visual effects.

Note that you must initialize a CNiImage object from an existing object. If you do not initialize a CNiImage object from an existing object, a CNiObjectNotInUsableState exception will be thrown when you attempt to manipulate the instance of the CNiImage.

 [Hierarchy Chart](#)

## ▲ Base Classes

---

### ▲ CNiInterface

## ◆ Data Items

---

- ◆ `long`      [AnimateColumns](#)      Specifies the number of columns in a bitmap that is being used for animation.
- ◆ `Speeds`      [AnimateInterval](#)      Specifies how often this image animates.
- ◆ `long`      [AnimateRows](#)      Specifies the number of rows in a bitmap that is being used for animation.
- ◆ `Speeds`      [BlinkInterval](#)      Specifies how often this image blinks.
- ◆ [CNiColor](#)      [Color](#)      Specifies the color for the image.
- ◆ `CNiPicture`      [Picture](#)      Specifies the image used in the `CNiImage` object.
  
- ◆ `bool`      [ReverseAnimation](#)      Specifies the direction of animation.
- ◆ `bool`      [SaveLink](#)      Specifies if the `CNiImage` object saves the image or a link to the image.
  
- ◆ `bool`      [Stretch](#)      Specifies if the image is displayed in its normal size or stretched to fit within the control.
  
- ◆ `bool`      [Tile](#)      Specifies if the image is tiled.
- ◆ `bool`      [Transparent](#)      Specifies if the image has a transparent color.
  
- ◆ [CNiColor](#)      [TransparentColor](#)      Specifies the color in the image that should be drawn as transparent.
  
- ◆ [CString](#)      [Url](#)      Specifies the path to the image.
- ◆ `bool`      [Visible](#)      Specifies if the image is visible.

## ◆ Constructors

---

- ◆ **CNiImage**() Default constructor.
- ◆ **CNiImage**( CWImage\_CI\* pCustom, [CNiInterface::ThreadAccess](#) option ) Constructor that attaches to the specified CWImage\_CI pointer.
- ◆ **CNiImage**( [const](#) CNiImage& source ) Copy constructor.

## ◆ **D e s t r u c t o r s**

---

◆ `~CNiImage()`

Destructor.

## ◆ **F u n c t i o n s**

---

- ◆ `static const IID & GetIid()` Returns the globally unique identifier (GUID) of the ActiveX interface to which this class connects.
- ◆ `const CNiImage & operator =(  
const  
CNiImage&  
source )` Assignment operator.
- ◆ `void Reload()` Loads the image referenced by the current value of the URL property.

## Example

---

```
// Change the image displayed in the plot area of a 2D graph and  
// set it to blink quickly.
```

```
CNiGraph3D graph;  
CNiImage image = graph.Images("Plot Area");  
image.URL = "c:\\windows\\circles.bmp";  
image.BlinkInterval = CNiImage::SpeedVeryFast;
```

---

# **CNiKnob**



**Class**

Declared in:  
**NiKnob.h**





## Overview

---

CNiKnob encapsulates the interface to the Measurement Studio ActiveX knob control, which represents different types of circular displays.

CNiKnob is capable of responding to events that are generated by the control. For a list of the events that can be generated by this control and details on how to respond to the events in your program, refer to the [Knob Events](#) page.

### Features

- Different display styles - dials, gauges, and meters.
- Ability to use the CniAxis class to interface to a single axis of a knob control. This allows you to modify the appearance and behavior of the axis.
- Automatic axis labeling with numeric scales (log or inverted) and values (continuous or discrete).
- Multiple pointers, each one representing one scalar value. A pointer indicates the current value of the knob.
- Custom ticks, labels, and value pairs. Ticks are the divisions that represent increments on the knob. Labels display the value of each tick. Value pairs are names paired with a value. For example, use a value pair to add the text label "Boiling Point" to a numeric axis at the value 212 F.
- CniKnob includes bindable properties. You can bind these properties to a DataSocket source or target. This allows you to read property values from and write property values to the source or target. Click [here](#) for a list of bindable properties.

Note: To specify a date/time value, you must convert your date or time value to a double. A date is implemented as a floating-point value with the integer part of the number measuring days from midnight, 30 December 1899, and the fractional part representing the time of day. The absolute value of the fractional part of the number represents the time as a fraction of a day. Thus, 1 second equals  $1 / 24 \text{ hours} / 60 \text{ minutes}$ , which is  $1/86400$  or approximately  $1.157407e-5$ . So, midnight, 31 December 1899, is represented by 1.0. Similarly, 6 AM, 1 January 1900, is represented by 2.25, and midnight, 29 December 1899, is -1.0. However, 6 AM, 29 December 1899, is -1.25.

 [Hierarchy Chart](#)

## ▲ Base Classes

---

### ▲ CNiControl

## ◆ Data Items

---

◆ <a href="#">CNIPointer</a>	<a href="#"><b>ActivePointer</b></a>	Specifies the CNIPointer object of the active pointer.
◆ double	<a href="#"><b>ArcEnd</b></a>	Specifies the end angle (in degrees) for the CNIKnob's axis arc.
◆ double	<a href="#"><b>ArcStart</b></a>	Specifies the start angle (in degrees) for the CNIKnob's axis arc.
◆ <a href="#">CNIAxis</a>	<a href="#"><b>Axis</b></a>	Returns the CNIAxis object for this control.
◆ <a href="#">CNIColor</a>	<a href="#"><b>BackColor</b></a>	Specifies the background color of the control.
◆ <a href="#">CNIBindings</a>	<a href="#"><b>Bindings</b></a>	Gets a collection of binding objects associated with the control.
◆ <a href="#">CString</a>	<a href="#"><b>Caption</b></a>	Specifies the text that appears in the control.
◆ <a href="#">CNIColor</a>	<a href="#"><b>CaptionColor</b></a>	Specifies the color of the caption.
◆ <a href="#">CNIControlMetrics</a>	<a href="#"><b>ControlMetrics</b></a>	Returns the ControlMetrics object for this control.
◆ bool	<a href="#"><b>Enabled</b></a>	Specifies if the control responds to user input.
◆ <a href="#">CNIFont</a>	<a href="#"><b>Font</b></a>	Specifies the font for the caption and axis labels.
◆ <a href="#">CNIColor</a>	<a href="#"><b>ForeColor</b></a>	Specifies the color of many objects in the CNIslide or CNIKnob control.
◆ bool	<a href="#"><b>ImmediateUpdates</b></a>	Specifies if the control draws new data as soon as it is available or if the form refreshes the control when it draws other controls.
◆ double	<a href="#"><b>IncDecValue</b></a>	Specifies the amount that the

◆ KeyboardModes	<b><u>KeyboardMode</u></b>	knob is changed when the user uses the keyboard to increment or decrement it. Specifies how the control handles keyboard input from the user.
◆ <u>CNiPointers</u>	<b><u>Pointers</u></b>	Gets a collection of pointer objects associated with the control.
◆ long	<b><u>ReadyState</u></b>	Returns the ready state of the control.
◆ ShowFocusModes	<b><u>ShowFocusMode</u></b>	Specifies how the control indicates that it has the focus.
◆ double	<b><u>Value</u></b>	Specifies the value of the active pointer.
◆ long	<b><u>ValuePairIndex</u></b>	Specifies the index of the value pair selected by the active pointer.
◆ bool	<b><u>Windowless</u></b>	Specifies if the control has a window.

## ◆ Constructors

---

◆ **CNiKnob**( [cNiInterface::ThreadAccess](#) option = [cNiInterface::MultipleThreadsWithCaching](#) )

## ◆ **D e s t r u c t o r s**

---

◆ ~CNiKnob()

## ◆ Functions

---

◆ void	<a href="#"><u>AboutBox()</u></a>	Displays the About Box for the control.
◆ CNIPicture	<a href="#"><u>ControlImage()</u></a>	Returns an image of the entire control.
◆ virtual BOOL	<a href="#"><u>Create</u></a> ( LPCTSTR lpszClassName, LPCTSTR lpszWindowName, DWORD dwStyle, <b>const</b> RECT& rect, <a href="#"><u>CWnd*</u></a> pParentWnd, UINT nID, CCreateContext* pContext = NULL )	Creates the ActiveX control that is represented in the MFC program by this object.
◆ BOOL	<a href="#"><u>Create</u></a> ( LPCTSTR lpszWindowName, DWORD dwStyle, <b>const</b> RECT& rect, <a href="#"><u>CWnd*</u></a> pParentWnd, UINT nID, CFile* pPersist = NULL, BOOL bStorage = FALSE, BSTR bstrLicKey = NULL )	Creates the ActiveX control that is represented in the MFC program by this object.
◆ BOOL	<a href="#"><u>CreateControl</u></a> ( LPCTSTR lpszClass, LPCTSTR lpszWindowName, DWORD dwStyle, <b>const</b> RECT& rect, <a href="#"><u>CWnd*</u></a> pParentWnd, UINT nID, CFile* pPersist = NULL, BOOL bStorage = FALSE, BSTR bstrLicKey = NULL )	Creates the ActiveX control that is represented in the MFC program by this object.
◆ BOOL	<a href="#"><u>CreateControl</u></a> ( REFCLSID clsid, LPCTSTR lpszWindowName, DWORD dwStyle, <b>const</b> RECT& rect, <a href="#"><u>CWnd*</u></a> pParentWnd, UINT nID, CFile* pPersist = NULL, BOOL bStorage = FALSE, BSTR bstrLicKey = NULL )	Creates the ActiveX control that is represented in the MFC program by this object.
◆ void	<a href="#"><u>ExportStyle</u></a> ( LPCTSTR fileName )	Exports the

◆ `static const CLSID & GetClsid()`

style of the control to a file.

Returns the globally unique identifier (GUID) of the ActiveX control to which this class connects.

◆ `static const IID & GetIid()`

Returns the globally unique identifier (GUID) of the ActiveX interface to which this class connects.

◆ `CNiImage Images( long imageIndex )`

Gets the image object associated with the specified part of the CNiKnob control.

◆ `CNiImage Images( const CString& imageName )`

Gets the image object associated with the specified part of the CNiKnob control.

◆ `void ImportStyle( LPCTSTR fileName )`

Imports a previously exported style.

◆ `void Refresh()`

Forces the control to redraw.

◆ `void SetBuiltinStyle( KnobStyles style )`

Sets many properties of the

 void

ValidateControl()

control to  
represent the  
new style  
specified.

Validates the  
current state of  
the control.

---

# **CNiLabels**



**Class**

Declared in:  
**NiLabels.h**





## Overview

---

CNiLabels encapsulates the interface to the Labels property of the CNiAxis object. You use labels to annotate the axes on controls such as graphs, slides, and knobs.

 [Hierarchy Chart](#)

## ▲ Base Classes

---

### ▲ CNiInterface

## ◆ Data Items

---

- ◆ `bool`      **Above** Specifies if labels appear above the object.
- ◆ `bool`      **Below** Specifies if labels appear below the object.
- ◆ `CNiColor`   **Color** Specifies the color of the labels.
- ◆ `bool`      **Left** Specifies if labels appear to the left of the object.
- ◆ `bool`      **Radial** Specifies if the system draws radial labels
- ◆ `bool`      **Right** Specifies if labels appear to the right of the object.
- ◆ `long`      **Width** Specifies the width of a label on the axis.

## ◆ Constructors

---

- ◆ **CNiLabels()** Default constructor.
- ◆ **CNiLabels**( CWLabels\_CI\* pCustom, **CNiInterface::ThreadAccess** option ) Constructor that attaches to the specified CWLabels\_CI pointer.
- ◆ **CNiLabels**( **const** CNiLabels& source ) Copy constructor.

## ◆ **D e s t r u c t o r s**

---

◆ `~CNiLabels()`

Destructor.

## ◆ **F u n c t i o n s**

---

- ◆ `static const IID & GetIid()` Returns the globally unique identifier (GUID) of the ActiveX interface to which this class connects.
  - ◆ `const CNiLabels & operator =(  
const  
CNiLabels&  
source )` Assignment operator.
-

# **CNiNumEdit**



**Class**

Declared in:  
**NiNumedit.h**





## Overview

---

CNiNumEdit encapsulates the interface to a Measurement Studio ActiveX numeric edit control, which displays and facilitates the editing of numeric values.

CNiNumEdit is capable of responding to events that are generated by the control. For a list of the events that can be generated by this control and details on how to respond to the events in your program, refer to the [Numeric Edit Events](#) page.

### Features

- Range checking.
- Increment and decrement buttons.
- Control or indicator display style. A control accepts input from users, while an indicator displays only its current value.
- Built-in numeric format styles, including scientific, symbolic engineering, scaling, time, and date.
- CNiNumEdit includes bindable properties. You can bind these properties to a DataSocket source or target. This allows you to read property values from and write property values to the source or target. Click [here](#) for a list of bindable properties.

Note: To specify a date/time value, you must convert your date or time value to a double. A date is implemented as a floating-point value with the integer part of the number measuring days from midnight, 30 December 1899, and the fractional part representing the time of day. The absolute value of the fractional part of the number represents the time as a fraction of a day. Thus, 1 second equals  $1 / 24 \text{ hours} / 60 \text{ minutes}$ , which is  $1/86400$  or approximately  $1.157407\text{e-}5$ . So, midnight, 31 December 1899, is represented by 1.0. Similarly, 6 AM, 1 January 1900, is represented by 2.25, and midnight, 29 December 1899, is -1.0. However, 6 AM, 29 December 1899, is -1.25.

 [Hierarchy Chart](#)

## ▲ Base Classes

---

### ▲ CNiControl

## ◆ Data Items

---

◆ double	<u>AccelInc</u>	Specifies the amount the value increments or decrements when the user holds down the increment or decrement button longer than the time specified by the AccelTime property.
◆ long	<u>AccelTime</u>	Specifies the number of seconds the increment or decrement button must be held down until the value is changed by the AccelInc value instead of the IncDecVal.
◆ Alignments	<u>Alignment</u>	Specifies how the text within the control is aligned.
◆ Appearances	<u>Appearance</u>	Specifies how the edit box portion of the control is drawn.
◆ <u>CNiColor</u>	<u>BackColor</u>	Specifies the background color of the edit box portion of the control.
◆ <u>CNiColor</u>	<u>BackgroundColor</u>	Specifies the background color of the increment and decrement buttons portion of the control.
◆ <u>CNiBindings</u>	<u>Bindings</u>	Gets a collection of binding objects associated with the control.
◆ BorderStyles	<u>BorderStyle</u>	Specifies the border around the edit box portion of the control.
◆ <u>CNiColor</u>	<u>ButtonColor</u>	Specifies the color of the increment and decrement buttons.
◆ ButtonStyles	<u>ButtonStyle</u>	Specifies the style of the increment and decrement buttons.
◆ bool	<u>Discrete</u>	Specifies if the control is in discrete or continuous mode.

◆ double	<b><u>DiscreteBase</u></b>	Specifies the interval to use in a discrete control.
◆ double	<b><u>DiscreteInterval</u></b>	Specifies the interval to use in a discrete control.
◆ bool	<b><u>Enabled</u></b>	Specifies if the user is able to interact with the control.
◆ <u>CNiFont</u>	<b><u>Font</u></b>	Specifies the font the control uses.
◆ <u>CString</u>	<b><u>FormatString</u></b>	Specifies the string that formats the value in the numeric edit control.
◆ Positions	<b><u>IncDecButtonPosition</u></b>	Specifies the location of the increment and decrement buttons on the control.
◆ bool	<b><u>IncDecButtonVisible</u></b>	Specifies if the increment and decrement buttons are visible.
◆ double	<b><u>IncDecValue</u></b>	Specifies the amount the value increments or decrements when the user clicks the increment or decrement button.
◆ double	<b><u>Maximum</u></b>	Specifies the maximum value of the control for range checking.
◆ double	<b><u>Minimum</u></b>	Specifies the minimum value of the control for range checking.
◆ NumEditModes	<b><u>Mode</u></b>	Specifies how the control operates.
◆ bool	<b><u>RangeChecking</u></b>	Specifies if the control enforces the minimum and maximum.
◆ <u>CString</u>	<b><u>Text</u></b>	Specifies the text that is displayed in the edit box.
◆ <u>CNiColor</u>	<b><u>TextColor</u></b>	Specifies the color of the text in the control.
◆ double	<b><u>Value</u></b>	Current value of the numeric edit control.

## ◆ Constructors

---

◆ **CNiNumEdit**( [CNiInterface::ThreadAccess](#) option = [CNiInterface::MultipleThreadsWithCaching](#) )

## ◆ **D e s t r u c t o r s**

---

◆ ~CNiNumEdit()

## ◆ Functions

---

◆ void	<a href="#"><u>AboutBox()</u></a>	Displays the About Box for the control.
◆ CNIPicture	<a href="#"><u>ControlImage()</u></a>	Returns an image of the entire control.
◆ virtual BOOL	<a href="#"><u>Create</u></a> ( LPCTSTR lpszClassName, LPCTSTR lpszWindowName, DWORD dwStyle, <b>const</b> RECT& rect, <a href="#"><u>CWnd*</u></a> pParentWnd, UINT nID, CCreateContext* pContext = NULL )	Creates the ActiveX control that is represented in the MFC program by this object.
◆ BOOL	<a href="#"><u>Create</u></a> ( LPCTSTR lpszWindowName, DWORD dwStyle, <b>const</b> RECT& rect, <a href="#"><u>CWnd*</u></a> pParentWnd, UINT nID, CFile* pPersist = NULL, BOOL bStorage = FALSE, BSTR bstrLicKey = NULL )	Creates the ActiveX control that is represented in the MFC program by this object.
◆ BOOL	<a href="#"><u>CreateControl</u></a> ( LPCTSTR lpszClass, LPCTSTR lpszWindowName, DWORD dwStyle, <b>const</b> RECT& rect, <a href="#"><u>CWnd*</u></a> pParentWnd, UINT nID, CFile* pPersist = NULL, BOOL bStorage = FALSE, BSTR bstrLicKey = NULL )	Creates the ActiveX control that is represented in the MFC program by this object.
◆ BOOL	<a href="#"><u>CreateControl</u></a> ( REFCLSID clsid, LPCTSTR lpszWindowName, DWORD dwStyle, <b>const</b> RECT& rect, <a href="#"><u>CWnd*</u></a> pParentWnd, UINT nID, CFile* pPersist = NULL, BOOL bStorage = FALSE, BSTR bstrLicKey = NULL )	Creates the ActiveX control that is represented in the MFC program by this object.
◆ void	<a href="#"><u>ExportStyle</u></a> ( LPCTSTR fileName )	Exports the

◆ static const CLSID & GetClsid()

style of the control to a file.

Returns the globally unique identifier (GUID) of the ActiveX control to which this class connects.

◆ static const IID & GetIid()

Returns the globally unique identifier (GUID) of the ActiveX interface to which this class connects.

◆ void ImportStyle( LPCTSTR fileName )

Imports a previously exported style.

◆ void SetMinMax( double minimum, double maximum )

Sets the Minimum and Maximum properties.

◆ void ValidateControl()

Validates the current state of the control.

---

# **CNiPlot**



**Class**

Declared in:  
**NiPlot.h**





## Overview

---

CNiPlot encapsulates the interface to a single plot of a graph control, which allows you to modify its appearance and behavior.

You get individual plots using the `Plots` property on a `CNiGraph` object.

Note: To specify a date/time value, you must convert your date or time value to a double. A date is implemented as a floating-point value with the integer part of the number measuring days from midnight, 30 December 1899, and the fractional part representing the time of day. The absolute value of the fractional part of the number represents the time as a fraction of a day. Thus, 1 second equals  $1 / 24 \text{ hours} / 60 \text{ minutes}$ , which is  $1/86400$  or approximately  $1.157407\text{e-}5$ . So, midnight, 31 December 1899, is represented by 1.0. Similarly, 6 AM, 1 January 1900, is represented by 2.25, and midnight, 29 December 1899, is -1.0. However, 6 AM, 29 December 1899, is -1.25.

 [Hierarchy Chart](#)

## ▲ Base Classes

---

### ▲ CNiInterface

## ◆ Data Items

---

◆ <code>bool</code>	<u><a href="#">AutoScale</a></u>	Specifies if the extents of the data in the plot affect the extents of an autoscaling axis.
◆ <code>CNiPlot</code>	<u><a href="#">BasePlot</a></u>	Specifies a plot to use for Y values when <code>FillToBase</code> or <code>LineToBase</code> is enabled.
◆ <code>double</code>	<u><a href="#">BaseValue</a></u>	Specifies the Y value to use when <code>FillToBase</code> or <code>LineToBase</code> is enabled.
◆ <code>double</code>	<u><a href="#">DefaultXFirst</a></u>	Specifies the default value used by the overloaded versions of the <code>PlotY</code> function that do not take the <code>xFirst</code> parameter.
◆ <code>double</code>	<u><a href="#">DefaultXInc</a></u>	Specifies the default value used by the overloaded versions of the <code>PlotY</code> and <code>ChartY</code> functions that do not take the <code>xInc</code> parameter.
◆ <code>bool</code>	<u><a href="#">DefaultXInFirstRow</a></u>	Specifies the default value used by the overloaded versions of the <code>PlotXY</code> and <code>ChartXY</code> functions that do not take the <code>xInFirstRow</code> parameter.
◆ <code>bool</code>	<u><a href="#">Enabled</a></u>	Specifies if the plot generates mouse events when <code>CNiGraph::TrackMode = CNiGraph::TrackAllEvents</code> and the plot is visible.
◆ <u><a href="#">CNiColor</a></u>	<u><a href="#">FillColor</a></u>	Specifies the color to use for filling to the <code>BaseValue</code> or <code>BasePlot</code> .
◆ <code>bool</code>	<u><a href="#">FillToBase</a></u>	Specifies if the function fills the area between the plot and the <code>CNiPlot::BaseValue</code> or <code>CNiPlot::BasePlot</code> .
◆ <u><a href="#">CNiColor</a></u>	<u><a href="#">LineColor</a></u>	Specifies the color of line that connects points in the plot.
◆ <code>LineStyle</code>	<u><a href="#">LineStyle</a></u>	Specifies the line style for connecting points on a plot.

◆

bool	<b><u>LineToBase</u></b>	Specifies if lines connect the data points to the <code>CNiPlot::BaseValue</code> or <code>CNiPlot::BasePlot</code> .
◆ <code>CNiColor</code>	<b><u>LineToBaseColor</u></b>	Specifies the color of lines connecting data points to the <code>CNiPlot::BaseValue</code> or <code>CNiPlot::BasePlot</code> .
◆ short	<b><u>LineWidth</u></b>	Specifies the width of the plotting line.
◆ bool	<b><u>MultiPlot</u></b>	Determines if the plot or chart functions can use this plot.
◆ <code>CString</code>	<b><u>Name</u></b>	Specifies the name of the plot.
◆ <code>CNiColor</code>	<b><u>PointColor</u></b>	Specifies the color for points on a plot.
◆ PointStyles	<b><u>PointStyle</u></b>	Specifies the point style for drawing points on a plot.
◆ bool	<b><u>TempPlot</u></b>	Specifies if you can discard a plot if the next <code>CNiGraph</code> plot or chart function does not need it.
◆ bool	<b><u>Visible</u></b>	Specifies if the plot is visible or hidden.
◆ <code>CNiAxis</code>	<b><u>XAxis</u></b>	Gets the x axis for a plot.
◆ <code>CNiAxis</code>	<b><u>YAxis</u></b>	Specifies the y axis for a plot.

## ◆ Constructors

---

- ◆ **CNiPlot**() Default constructor.
- ◆ **CNiPlot**( CWPlot\_CI\* pCustom, **CNiInterface::ThreadAccess** option ) Constructor that attaches to the specified CWPlot\_CI pointer.
- ◆ **CNiPlot**( **const** CNiPlot& source ) Copy constructor.

## ◆ **D e s t r u c t o r s**

---

◆ `~CNiPlot()`

Destructor.

## ◆ Functions

---

- ◆ void **ChartXvsY**( const [CniVector](#)& x, const [CniVector](#)& y ) Charts a vector of Y data against a vector of X data.
- ◆ void **ChartXY**( const [CniMatrix](#)& xy, bool xInFirstRow ) Charts a matrix of data as an X-Y plot.
- ◆ void **ChartXY**( const [CniMatrix](#)& xy ) Charts a matrix of data as an X-Y plot.
- ◆ void **ChartXY**( double x, double y ) Charts a matrix of data as an X-Y plot.
- ◆ void **ChartY**( const [CniVector](#)& y, double xInc ) Charts a vector of data.
- ◆ void **ChartY**( const [CniVector](#)& y ) Charts a vector of data.
- ◆ void **ChartY**( double y, double xInc ) Charts a point of data.
- ◆ void **ChartY**( double y ) Charts a point of data.
- ◆ void **ClearData**() Clears the data currently displayed in the plot.
- ◆ static const IID & **GetIid**() Returns the globally unique identifier (GUID) of the ActiveX interface to which this class connects.
- ◆ const [CniPlot](#) & operator =( const [CniPlot](#)& source ) Assignment operator.
- ◆ void **PlotXvsY**( const [CniVector](#)& x, const [CniVector](#)& y ) Plots a vector of Y data against a vector of X data.

- ◆ void **PlotXY**( const CNiMatrix& xy, bool xInFirstRow ) Plots a matrix of data as an X-Y plot.
- ◆ void **PlotXY**( const CNiMatrix& xy ) Plots a matrix of data as an X-Y plot.
- ◆ void **PlotXY**( double x, double y ) Plots a single point of data as an X-Y plot.
- ◆ void **PlotY**( const CNiVector& y, double xFirst, double xInc ) Plots a vector of data.
- ◆ void **PlotY**( const CNiVector& y, double xFirst ) Plots a vector of data.
- ◆ void **PlotY**( const CNiVector& y ) Plots a vector of data.

## Example

---

```
// Add a sine wave plot to the graph.
```

```
CNiGraph graph;  
CNiReal64Vector wfm(1000);  
CNiMath::SineWave(wfm);  
graph.PlotY(wfm);
```

```
// Get the plot from the graph and change its line style.
```

```
CNiPlot plot = graph.Plots.Item(1);  
plot.LineStyle = CNiPlot::LineDash;
```

---

# **CNiPlots**



**Class**

Declared in:  
**NiPlots.h**





## Overview

---

CNiPlots encapsulates the interface to the Plots property of a CNiGraph object, which allows you to access and remove plots associated with the graph control.

- Use the `CNiGraph::Plots` property to obtain the plots collection for the graph.
- Use the `Add` function to create additional plots. `Add` returns a `CNiPlot` object, which represents the new plot.
- Use the `Item` function to access existing plots in the collection. This function can access plots by either name or index.
- Use the `Remove` function to remove existing plots from the collection. This function can access plots by either name or index.
- Use the `RemoveAll` function to remove all plots from the graph.

 [Hierarchy Chart](#)

## ▲ Base Classes

---

### ▲ CNiInterface

## ◆ Data Items

---

◆ [short](#)      [Count](#) Returns the number of plots in the collection.

## ◆ Constructors

---

- ◆ **CNiPlots()** Default constructor.
- ◆ **CNiPlots**( CWPlots\_CI\* pCustom, [CniInterface::ThreadAccess](#) option ) Constructor that attaches to the specified CWPlots\_CI pointer.
- ◆ **CNiPlots**( [const](#) CNiPlots& source ) Copy constructor.

## ◆ **D e s t r u c t o r s**

---

◆ `~CNiPlots()`

Destructor.

## ◆ **F u n c t i o n s**

---

- ◆ CNiPlot                    **Add()**                    Adds a plot to the collection and returns the new plot.
- ◆ `static const IID &` **GetIid()**                    Returns the globally unique identifier (GUID) of the ActiveX interface to which this class connects.
- ◆ CNiPlot                    **Item**( `long` plotIndex )                    Returns the specified plot from the collection.
- ◆ CNiPlot                    **Item**( `const CString&` plotName )                    Returns the specified plot from the collection.
- ◆ `const CNiPlots &` **operator =**( `const CNiPlots&` source )                    Assignment operator.
- ◆ `void`                    **Remove**( `long` plotIndex )                    Removes the specified plot from the collection.
- ◆ `void`                    **Remove**( `const CString&` plotName )                    Removes the specified plot from the collection.
- ◆ `void`                    **RemoveAll**()                    Removes all plots from the collection.

## Examples

---

1. Set the first plot in the graph to auto scale.

```
CNiGraph graph;  
CNiPlot plot = graph.Plots.Item(1);  
plot.AutoScale = true;
```

2. Set the point style on the first plot in the graph.

```
CNiGraph graph;  
graph.Plots.Item(1).PointStyle = CNiPlot::PointAsterisk;
```

3. Add a new plot to the graph.

```
CNiGraph graph;  
CNiPlot plot = graph.Plots.Add();  
plot.FillToBase = true;
```

4. Remove all plots from the graph.

```
CNiGraph graph;  
graph.Plots.RemoveAll();
```

5. Change the name of the first plot to "Velocity".

```
CNiGraph graph;  
graph.Plots.Item(1).Name = "Velocity";
```

Subsequent accesses to the plot must now use "Velocity" as the item name. For example,

```
graph.Plots.Item("Velocity").Visible = true;
```

---

# **CNiPointer**



**Class**

Declared in:  
**NiPointer.h**





## Overview

---

`CNiPointer` encapsulates the interface to a single pointer on a knob or slide control, which allows you to modify its appearance and behavior.

You get individual pointers using the `Pointers` property on a `CNiKnob` or `CNiSlide` object.

Note: To specify a date/time value, you must convert your date or time value to a double. A date is implemented as a floating-point value with the integer part of the number measuring days from midnight, 30 December 1899, and the fractional part representing the time of day. The absolute value of the fractional part of the number represents the time as a fraction of a day. Thus, 1 second equals  $1 / 24 \text{ hours} / 60 \text{ minutes}$ , which is  $1/86400$  or approximately  $1.157407\text{e-}5$ . So, midnight, 31 December 1899, is represented by 1.0. Similarly, 6 AM, 1 January 1900, is represented by 2.25, and midnight, 29 December 1899, is -1.0. However, 6 AM, 29 December 1899, is -1.25.

 [Hierarchy Chart](#)

## ▲ Base Classes

---

### ▲ CNiInterface

## ◆ Data Items

---

◆ <a href="#">CNiColor</a>	<b><u>Color</u></b>	Specifies the color of the pointer.
◆ <a href="#">CNiColor</a>	<b><u>FillColor</u></b>	Specifies the fill color of the pointer.
◆ PointerFillStyles	<b><u>FillStyle</u></b>	Specifies how the function fills the pointer.
◆ <a href="#">long</a>	<b><u>Index</u></b>	Specifies the index of this pointer in the <code>CNiPointers</code> collection.
◆ PointerModes	<b><u>Mode</u></b>	Specifies how the pointer behaves and responds to user input.
◆ <a href="#">CString</a>	<b><u>Name</u></b>	Specifies the name of the pointer.
◆ PointerStyles	<b><u>Style</u></b>	Specifies the graphical style of the pointer.
◆ <a href="#">double</a>	<b><u>Value</u></b>	Specifies the value of the pointer.
◆ <a href="#">long</a>	<b><u>ValuePairIndex</u></b>	Specifies the index of the value pair selected by this pointer.
◆ <a href="#">bool</a>	<b><u>Visible</u></b>	Specifies if the pointer is visible or hidden.

## ◆ Constructors

---

- ◆ **CNiPointer**() Default constructor.
- ◆ **CNiPointer**( CWPointer\_CI\* pCustom, [CNiInterface::ThreadAccess](#) option ) Constructor that attaches to the specified CWPointer\_CI pointer.
- ◆ **CNiPointer**( [const](#) CNiPointer& source ) Copy constructor.

## ◆ **D e s t r u c t o r s**

---

◆ `~CNIPointer()`

Destructor.

## ◆ **F u n c t i o n s**

---

- ◆ `static const IID & GetIid()` Returns the globally unique identifier (GUID) of the ActiveX interface to which this class connects.
- ◆ `const CNiPointer & operator =(  
const  
CNiPointer&  
source )` Assignment operator.

## Example

---

// Set the style of the first pointer in the knob control.

```
CNiKnob knob;  
CNiPointer pointer = knob.Pointers.Item(1);  
pointer.Style = CNiPointer::Pointer3D;
```

---

# **CNiPointers**



**Class**

Declared in:  
**NiPointers.h**





## Overview

---

A `CNiPointers` object is a collection of pointers on a knob or slide control.

- Use `CNiKnob::Pointers` to obtain the pointers collection for a knob.
- Use `CNiSlide::Pointers` to obtain the pointers collection for a slide.
- Use the `Add` function to create additional pointers. `Add` returns a `CNiPointer` object, which represents the new pointer.
- Use the `Item` function to access existing pointers in the collection. This function can access pointers by either name or index.
- Use the `Remove` function to remove existing pointers from the collection. This function can access pointers by either name or index.
- Use the `RemoveAll` function to remove all pointers from the collection.

 [Hierarchy Chart](#)

## ▲ Base Classes

---

### ▲ CNiInterface

## ◆ Data Items

---

◆ `short`     Count Returns the number of pointers in the collection.

## ◆ Constructors

---

- ◆ **CNiPointers()** Default constructor.
- ◆ **CNiPointers**( CWPointers\_CI\* pCustom, **CNiInterface::ThreadAccess** option ) Constructor that attaches to the specified CWPointers\_CI pointer.
- ◆ **CNiPointers**( **const** CNiPointers& source ) Copy constructor.

## ◆ **D e s t r u c t o r s**

---

◆ `~CNiPointers()`

Destructor.

## ◆ **F u n c t i o n s**

---

- ◆ CNiPointer            **Add()**            Adds a pointer to the collection and returns the new pointer.
- ◆ `static const IID &`    **GetId()**            Returns the globally unique identifier (GUID) of the ActiveX interface to which this class connects.
- ◆ CNiPointer            **Item**( long pointerIndex ) Returns the specified pointer from the collection.
- ◆ CNiPointer            **Item**( const CString& pointerName ) Returns the specified pointer from the collection.
- ◆ `const CNiPointers &` **operator =**( const CNiPointers& source )    Assignment operator.
- ◆ `void`                    **Remove**( long pointerIndex ) Removes the specified pointer from the collection.
- ◆ `void`                    **Remove**( const CString& pointerName ) Removes the specified pointer from the collection.
- ◆ `void`                    **RemoveAll**() Removes all pointers from the collection.

## ▣ Examples

---

1. Set the style of the first pointer in the knob control.

```
CNiKnob knob;  
CNiPointer pointer = knob.Pointers.Item(1);  
pointer.Style = CNiPointer::Pointer3D  
xAxis.Visible = true;
```

2. Set the current value of the pointer to 30.

```
CNiKnob knob;  
knob.Pointers.Item(1).Value = 30;
```

3. Add an additional pointer named "Power" to the slide control.

```
CNiSlide slide;  
slide.Pointers.Add("Power");
```

4. Remove all pointers from the slide control.

```
CNiSlide slide;  
slide.Pointers.RemoveAll();
```

5. Change the name of the first pointer to "Time".

```
CNiGraph slide;  
slide.Pointers.Item(1).Name = "Time";
```

Subsequent accesses to this pointer must now use "Time" as the item name. For example,

```
slide.Pointers.Item("Time").Visible = true;
```

---

# **CNiShape**



**Class**

Declared in:  
**NiShape.h**





## Overview

---

CNiShape encapsulates the interface to a single shape for an annotation on a graph control, which allows you to modify the appearance and behavior of the shape.

You get individual shapes using the Shape property on a CNiAnnotation object.

Note: To specify a date/time value, you must convert your date or time value to a double. A date is implemented as a floating-point value with the integer part of the number measuring days from midnight, 30 December 1899, and the fractional part representing the time of day. The absolute value of the fractional part of the number represents the time as a fraction of a day. Thus, 1 second equals  $1 / 24 \text{ hours} / 60 \text{ minutes}$ , which is  $1/86400$  or approximately  $1.157407e-5$ . So, midnight, 31 December 1899, is represented by 1.0. Similarly, 6 AM, 1 January 1900, is represented by 2.25, and midnight, 29 December 1899, is -1.0. However, 6 AM, 29 December 1899, is -1.25.

 [Hierarchy Chart](#)

## ▲ Base Classes

---

### ▲ CNiInterface

## ◆ Data Items

---

◆ <a href="#">CIColor</a>	<b><u>Color</u></b>	Specifies the color of the shape.
◆ <a href="#">bool</a>	<b><u>FillVisible</u></b>	Specifies if the shape is filled with <code>CNishape::Color</code> .
◆ <a href="#">CUIImage</a>	<b><u>Image</u></b>	Specifies the shape image.
◆ <a href="#">CIColor</a>	<b><u>LineColor</u></b>	Specifies the line color of the shape.
◆ <a href="#">LineStyle</a>	<b><u>LineStyle</u></b>	Specifies the line style of the shape.
◆ <a href="#">short</a>	<b><u>LineWidth</u></b>	Specifies the line width of the shape.
◆ <a href="#">PointStyles</a>	<b><u>PointStyle</u></b>	Specifies the point style for the shape.
◆ <a href="#">RegionAreas</a>	<b><u>RegionArea</u></b>	Specifies the region area of the shape.
◆ <a href="#">Types</a>	<b><u>Type</u></b>	Specifies the type of shape.
◆ <a href="#">CNIReal64Vector</a>	<b><u>XCoordinates</u></b>	Specifies the x coordinates of the shape.
◆ <a href="#">CNIReal64Vector</a>	<b><u>YCoordinates</u></b>	Specifies the y coordinates of the shape.

## ◆ Constructors

---

- ◆ **CNiShape**() Default constructor.
- ◆ **CNiShape**( CWSHape\_CI\* pCustom, [CNiInterface::ThreadAccess](#) option ) Constructor that attaches to the specified CWSHape\_CI pointer.
- ◆ **CNiShape**( const CNiShape& source ) Copy constructor.

## ◆ **D e s t r u c t o r s**

---

◆ `~CNiShape()`

Destructor.

## ◆ Functions

---

- ◆ `static const IID & GetIid()` Returns the globally unique identifier (GUID) of the ActiveX interface to which this class connects.
- ◆ `const CNiShape & operator =( const CNiShape& source )` Assignment operator.
- ◆ `void SetCoordinates( const CNiVector & xCoordinates, const CNiVector & yCoordinates )` Sets the x and y coordinates of the shape at the same time.

## Example

---

```
// Change the type of the shape on the first annotation.
```

```
CNiGraph graph;  
CNiShape shape = graph.Annotations.Item(1).Shape;  
shape.Type = CNiShape::Rectangle;
```

---

# **CNiSlide**



**Class**

Declared in:  
**NiSlide.h**





## Overview

---

CNiSlide encapsulates the interface to the Measurement Studio ActiveX slide control, which represents different types of linear displays.

CNiSlide is capable of responding to events that are generated by the control. For a list of the events that can be generated by this control and details on how to respond to the events in your program, refer to the [Slide Events](#) page.

### Features

- Different display styles - vertical and horizontal slides, tanks, and thermometers.
- Ability to use the CNiAxis class to interface to a single axis of a slide control. This allows you to modify the appearance and behavior of the axis.
- Automatic axis labeling with numeric scales (log or inverted) and values (continuous or discrete).
- Multiple pointers, each one representing one scalar value.
- Custom ticks, labels, and value pairs. Ticks are the divisions that represent values on the slide. Labels display the value of each tick. Value pairs are names paired with a value. For example, use a value pair to add the text label "Boiling Point" to the value 212 F.
- Built-in format styles, including scientific, symbolic engineering, scaling, time, and date, for the labels.
- Animation - you can animate different parts of the control. For example, you might want to animate a pointer if its value exceeds a safe limit in the application.
- Custom images - you can add images to different parts of the control, including the background and pointers.

CNiSlide includes bindable properties. You can bind these properties to a DataSocket source or target. This allows you to read property values from and write property values to the source or target. Click [here](#) for a list of bindable properties.

Note: To specify a date/time value, you must convert your date or time value to a double. A date is implemented as a floating-point value with the integer part of the number measuring days from midnight, 30 December 1899, and the fractional part representing the time of day. The absolute value of the fractional part of the number represents the time as a fraction of a day. Thus, 1 second

equals  $1 / 24 \text{ hours} / 60 \text{ minutes}$ , which is  $1/86400$  or approximately  $1.157407\text{e-}5$ . So, midnight, 31 December 1899, is represented by 1.0. Similarly, 6 AM, 1 January 1900, is represented by 2.25, and midnight, 29 December 1899, is -1.0. However, 6 AM, 29 December 1899, is -1.25.

 [Hierarchy Chart](#)

## ▲ Base Classes

---

### ▲ CNiControl

## ◆ Data Items

---

◆ <a href="#">CNIPointer</a>	<b><u>ActivePointer</u></b>	Specifies the CNIPointer object of the active pointer.
◆ <a href="#">CNIAxis</a>	<b><u>Axis</u></b>	Returns the CNIAxis object for this control.
◆ <a href="#">CNIColor</a>	<b><u>BackColor</u></b>	Specifies the background color of the control.
◆ <a href="#">CNIBindings</a>	<b><u>Bindings</u></b>	Gets a collection of binding objects associated with the control.
◆ <a href="#">CString</a>	<b><u>Caption</u></b>	Specifies the text that appears in the control.
◆ <a href="#">CNIColor</a>	<b><u>CaptionColor</u></b>	Specifies the color of the caption.
◆ <a href="#">CNIControlMetrics</a>	<b><u>ControlMetrics</u></b>	Returns the CNIControlMetrics object associated with this control.
◆ <a href="#">bool</a>	<b><u>Enabled</u></b>	Specifies if the control responds to user input.
◆ <a href="#">CNIFont</a>	<b><u>Font</u></b>	Specifies the font for the caption and axis labels.
◆ <a href="#">CNIColor</a>	<b><u>ForeColor</u></b>	Specifies the color of many objects in the CNIslide or CNIKnob control.
◆ <a href="#">bool</a>	<b><u>ImmediateUpdates</u></b>	Specifies if the control draws new data as soon as it is available or if the form refreshes the control when it draws other controls.
◆ <a href="#">double</a>	<b><u>IncDecValue</u></b>	Specifies the value by which the pointer value is incremented/decremented when you click on the increment or decrement buttons use the keyboard to increment or decrement the

◆ <u>CNiColor</u>	<u><b>InteriorColor</b></u>	pointer. Specifies the color used to paint the interior of several types of slide styles.
◆ KeyboardModes	<u><b>KeyboardMode</b></u>	Specifies how the control handles keyboard input from the user.
◆ <u>CNiPointers</u>	<u><b>Pointers</b></u>	Gets a collection of pointer objects associated with the control.
◆ long	<u><b>ReadyState</b></u>	Returns the ready state of the control.
◆ ShowFocusModes	<u><b>ShowFocusMode</b></u>	Specifies how the control indicates that it has the focus.
◆ double	<u><b>Value</b></u>	Specifies the value of the active pointer.
◆ long	<u><b>ValuePairIndex</b></u>	Specifies the index of the value pair selected by the active pointer.
◆ bool	<u><b>Windowless</b></u>	Specifies if the control has a window.

## ◆ Constructors

---

◆ **CNiSlide**( [CNiInterface::ThreadAccess](#) option = [CNiInterface::MultipleThreadsWithCaching](#) )

## ◆ **D e s t r u c t o r s**

---

◆ ~C NiSlide()

## ◆ Functions

---

◆ void	<a href="#"><u>AboutBox()</u></a>	Displays the About Box for the control.
◆ CNIPicture	<a href="#"><u>ControlImage()</u></a>	Returns an image of the entire control.
◆ virtual BOOL	<a href="#"><u>Create</u></a> ( LPCTSTR lpszClassName, LPCTSTR lpszWindowName, DWORD dwStyle, <b>const</b> RECT& rect, <a href="#"><u>CWnd*</u></a> pParentWnd, UINT nID, CCreateContext* pContext = NULL )	Creates the ActiveX control that is represented in the MFC program by this object.
◆ BOOL	<a href="#"><u>Create</u></a> ( LPCTSTR lpszWindowName, DWORD dwStyle, <b>const</b> RECT& rect, <a href="#"><u>CWnd*</u></a> pParentWnd, UINT nID, CFile* pPersist = NULL, BOOL bStorage = FALSE, BSTR bstrLicKey = NULL )	Creates the ActiveX control that is represented in the MFC program by this object.
◆ BOOL	<a href="#"><u>CreateControl</u></a> ( LPCTSTR lpszClass, LPCTSTR lpszWindowName, DWORD dwStyle, <b>const</b> RECT& rect, <a href="#"><u>CWnd*</u></a> pParentWnd, UINT nID, CFile* pPersist = NULL, BOOL bStorage = FALSE, BSTR bstrLicKey = NULL )	Creates the ActiveX control that is represented in the MFC program by this object.
◆ BOOL	<a href="#"><u>CreateControl</u></a> ( REFCLSID clsid, LPCTSTR lpszWindowName, DWORD dwStyle, <b>const</b> RECT& rect, <a href="#"><u>CWnd*</u></a> pParentWnd, UINT nID, CFile* pPersist = NULL, BOOL bStorage = FALSE, BSTR bstrLicKey = NULL )	Creates the ActiveX control that is represented in the MFC program by this object.
◆ void	<a href="#"><u>ExportStyle</u></a> ( LPCTSTR fileName )	Exports the

◆ `static const CLSID & GetClsid()`

style of the control to a file.

Returns the globally unique identifier (GUID) of the ActiveX control to which this class connects.

◆ `static const IID & GetIid()`

Returns the globally unique identifier (GUID) of the ActiveX interface to which this class connects.

◆ `CNiImage Images( long itemIndex )`

Gets the image object associated with the specified part of the `CNiSlide` control.

◆ `CNiImage Images( const CString& itemName )`

Gets the image object associated with the specified part of the `CNiSlide` control.

◆ `void ImportStyle( LPCTSTR fileName )`

Imports a previously exported style.

◆ `void Refresh()`

Forces the control to redraw.

◆ `void SetBuiltinStyle( SlideStyles style )`

Sets many

 void

**ValidateControl()**

properties of the control to represent the new style specified.

Validates the current state of the control.

---

# **CNiTicks**



**Class**

Declared in:  
**NiTicks.h**





## Overview

---

CNiTicks encapsulates the interface to the `Ticks` property of a `CNiAxis` object, which allows you to modify the appearance and behavior of the tick marks on an axis.

Note: To specify a date/time value, you must convert your date or time value to a double. A date is implemented as a floating-point value with the integer part of the number measuring days from midnight, 30 December 1899, and the fractional part representing the time of day. The absolute value of the fractional part of the number represents the time as a fraction of a day. Thus, 1 second equals  $1 / 24 \text{ hours} / 60 \text{ minutes}$ , which is  $1/86400$  or approximately  $1.157407\text{e-}5$ . So, midnight, 31 December 1899, is represented by 1.0. Similarly, 6 AM, 1 January 1900, is represented by 2.25, and midnight, 29 December 1899, is -1.0. However, 6 AM, 29 December 1899, is -1.25.

 [Hierarchy Chart](#)

## ▲ Base Classes

---

### ▲ CNiInterface

## ◆ Data Items

---

◆ bool	<u>Above</u>	Specifies if ticks are drawn above the object.
◆ bool	<u>AutoDivisions</u>	Specifies if divisions are automatically calculated.
◆ bool	<u>Below</u>	Specifies if ticks are drawn below the object.
◆ bool	<u>Inside</u>	Specifies if tick marks are drawn on the inside of the axis.
◆ bool	<u>Left</u>	Specifies if tick marks are drawn to the left of the object.
◆ double	<u>MajorDivisions</u>	Specifies the number of major divisions.
◆ bool	<u>MajorGrid</u>	Specifies if major grid lines are drawn.
◆ <u>CNiColor</u>	<u>MajorGridColor</u>	Specifies the color of major grid lines.
◆ <u>CNiColor</u>	<u>MajorTickColor</u>	Specifies the color of major grid ticks.
◆ bool	<u>MajorTicks</u>	Specifies if major grid ticks are drawn.
◆ double	<u>MajorUnitsBase</u>	Specifies the base number for calculating ticks.
◆ double	<u>MajorUnitsInterval</u>	Specifies the number of units between major divisions.
◆ double	<u>MinorDivisions</u>	Specifies the number of minor divisions for each major division.
◆ bool	<u>MinorGrid</u>	Specifies if minor grid lines are drawn.
◆ <u>CNiColor</u>	<u>MinorGridColor</u>	Specifies the color of minor grid lines.
◆ <u>CNiColor</u>	<u>MinorTickColor</u>	Specifies the color of minor grid ticks.
◆ bool	<u>MinorTicks</u>	Specifies if minor grid ticks are drawn.
◆ double	<u>MinorUnitsInterval</u>	Specifies the number of units between minor divisions.
◆ bool	<u>Outside</u>	Specifies if tick marks are drawn on the outside of the axis
◆ bool	<u>Right</u>	Specifies if tick marks are drawn to the right of the object.

## ◆ Constructors

---

- ◆ **CNiTicks()** Default constructor.
- ◆ **CNiTicks**( CWTicks\_CI\* pCustom, [CniInterface::ThreadAccess](#) option ) Constructor that attaches to the specified cwticks\_CI pointer.
- ◆ **CNiTicks**( [const](#) CNiTicks& source ) Copy constructor.

## ◆ **D e s t r u c t o r s**

---

◆ `~CNiTicks()`

Destructor.

## ◆ **F u n c t i o n s**

---

- ◆ `static const IID & GetIid()` Returns the globally unique identifier (GUID) of the ActiveX interface to which this class connects.
- ◆ `const CNiTicks & operator =(  
const  
CNiTicks&  
source )` Assignment operator.

## Example

---

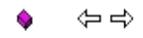
```
// Changes the number of major divisions on the x axis of a graph  
// control.
```

```
CNiGraph graph;  
CNiTicks ticks = graph.Axes.Item("XAxis").Ticks;  
ticks.MajorDivisions = 20;
```

---

# **Button Events**

**Function Group**



Declared in:  
**NiButtonEvents.h**





## Overview

---

The Measurement Studio button control generates a variety of events in response to user input or changes in properties that are bound to a DataSocket source.

Complete the following steps to add an event handler for a particular event generated by the button control.

1. Edit a dialog resource and add a Measurement Studio button control to the dialog.
2. Right-click on the button control and select **Events**.
3. Select the appropriate event to handle and select **Add and Edit**. By default, this creates a new member function, which is named in the form shown below.

On[eventName][objectIdSuffix]

To help you find the appropriate online help, the components of the member function name are described below.

- **eventName** is the event name.
- **objectIdSuffix** is the last part of the ID assigned to the control.

For example, if you have a button control for which the ID is IDC\_CWBUTTON1 and you want to add a handler for the `click` event. The default member function name created by the ClassWizard is `OnClickCwbutton1`.

4. Click **OK** to add and edit the new event handler. Notice that the prototype of the member function matches the corresponding event listed below. You can select this link to get additional information regarding the event.

Note: Various button events pass COM VARIANT data types as parameters to the event handler. Measurement Studio includes the `CnVariant` class, which provides a convenient interface for managing these VARIANT data types. Refer to the documentation for each event for further information.

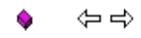
## ◆ Functions

---

- ◆ void [OnClick\(\)](#) Generated when you click the mouse on the control.
  - ◆ void [OnCWBindingDataUpdated](#)( short Index, LPDISPATCH Data, BOOL FAR\* Ignore ) Generated when the binding data is updated.
  - ◆ void [OnCWBindingStatusUpdated](#)( short Index, long Status, long Error, LPCTSTR Message ) Generated when the status of the binding connection changes.
  - ◆ void [OnDblClick\(\)](#) Generated when you double-click the mouse on the control.
  - ◆ void [OnKeyDown](#)( short FAR\* KeyCode, short Shift ) Generated when you press a key while the control has the input focus.
  - ◆ void [OnKeyPress](#)( short FAR\* KeyAscii ) Generated when a KeyDown message generates a key while a control is active.
  - ◆ void [OnKeyUp](#)( short FAR\* KeyCode, short Shift ) Generated when you release a key while the control has the input focus.
  - ◆ void [OnMouseDown](#)( short Button, short Shift, long x, long y ) Generated when you click the mouse on the control.
  - ◆ void [OnMouseMove](#)( short Button, short Shift, long x, long y ) Generated when you move the mouse over the control.
  - ◆ void [OnMouseUp](#)( short Button, short Shift, long x, long y ) Generated when you release the mouse on the control.
  - ◆ void [OnReadyStateChange](#)() Generated when the ready state changes.
  - ◆ void [OnValueChanged](#)( BOOL Value ) Generated when the value of the button changes.
-

# **Graph Events**

**Function Group**



Declared in:  
**NiGraphEvents.h**





## Overview

---

The Measurement Studio graph control generates a variety of events in response to user input or changes in properties that are bound to a DataSocket source.

Complete the following steps to add an event handler for a particular event generated by the graph control.

1. Edit a dialog resource and add a Measurement Studio graph control to the dialog.
2. Right-click on the graph control and select **Events**.
3. Select the appropriate event to handle and select **Add and Edit**. By default, this creates a new member function for which the name has the form shown below.

On[eventName][objectIdSuffix]

To help you find the appropriate online help, the components of the member function name are described below.

- **eventName** is the event name.
- **objectIdSuffix** is the last part of the ID assigned to the control.

For example, if you have a graph control for which the ID is IDC\_CWGRAPH1 and you want to add a handler for the `Click` event, the default member function name created by the ClassWizard is `OnClickCwgraph1`.

4. Click **OK** to add and edit the new event handler. Notice that the prototype of the member function matches the corresponding event listed below. You can select this link to get additional information regarding the event.

Note: Various graph events pass COM VARIANT data types as parameters to the event handler. Measurement Studio includes the `CNiVariant` class, which provides a convenient interface for managing these VARIANT data types. Refer to the documentation for each event for further information.

## ◆ Functions

---

- ◆ **void OnAnnotationChange**( long FAR\* AnnotationIndex, long FAR\* AnnotationPart, BOOL FAR\* bTracking ) Generated when you reposition an annotation with the mouse.
- ◆ **void OnAnnotationMouseDown**( short FAR\* Button, short FAR\* Shift, short FAR\* AnnotationIndex, long FAR\* AnnotationPart ) Generated when you click the mouse on an annotation.
- ◆ **void OnAnnotationMouseMove**( short FAR\* Button, short FAR\* Shift, short FAR\* AnnotationIndex, long FAR\* AnnotationPart ) Generated when you move the mouse over an annotation.
- ◆ **void OnAnnotationMouseUp**( short FAR\* Button, short FAR\* Shift, short FAR\* AnnotationIndex, long FAR\* AnnotationPart ) Generated when you release the mouse over an annotation.
- ◆ **void OnClick**() Generated when you click the mouse on the control.
- ◆ **void OnCursorChange**( long FAR\* CursorIndex, VARIANT FAR\* XPos, VARIANT FAR\* YPos, BOOL FAR\* bTracking ) Generated when you reposition a cursor with the mouse.
- ◆ **void OnCursorMouseDown**( short FAR\* Button, short FAR\* Shift, VARIANT FAR\* XPos, VARIANT FAR\* YPos, short FAR\* CursorIndex, long FAR\* CursorPart ) Generated when you click the mouse on a cursor.
- ◆ **void OnCursorMouseMove**( short FAR\* Button, short FAR\* Shift, VARIANT FAR\* XPos, VARIANT FAR\* YPos, short FAR\* CursorIndex, long FAR\* CursorPart ) Generated when you move the mouse over a cursor.
- ◆ **void OnCursorMouseUp**( short FAR\* Button, short FAR\* Shift, VARIANT FAR\* XPos, VARIANT FAR\* YPos, short FAR\* ) Generated when you release the mouse over a cursor.

CursorIndex, long FAR\* CursorPart )

- ◆ void **OnCWBindingDataUpdated**( short Index, LPDISPATCH Data, BOOL FAR\* Ignore ) Generated when the binding data is updated.
- ◆ void **OnCWBindingStatusUpdated**( short Index, long Status, long Error, LPCTSTR Message ) Generated when the status of the binding connection changes.
- ◆ void **OnDbClick**() Generated when you double-click the mouse on the control.
- ◆ void **OnKeyDown**( short FAR\* KeyCode, short Shift ) Generated when you press a key while the control has the input focus.
- ◆ void **OnKeyPress**( short FAR\* KeyAscii ) Generated when a KeyDown message generates a key while a control is active.
- ◆ void **OnKeyUp**( short FAR\* KeyCode, short Shift ) Generated when you release a key while the control has the input focus.
- ◆ void **OnMouseDown**( short Button, short Shift, long x, long y ) Generated when you click the mouse on the control.
- ◆ void **OnMouseMove**( short Button, short Shift, long x, long y ) Generated when you move the mouse over the control.
- ◆ void **OnMouseUp**( short Button, short Shift, long x, long y ) Generated when you release the mouse on the control.
- ◆ void **OnPlotAreaMouseDown**( short FAR\* Button, short FAR\* Shift, VARIANT FAR\* XPos, VARIANT FAR\* YPos ) Generated when you click the mouse on the plot area.
- ◆ void **OnPlotAreaMouseMove**( short FAR\* Button, short FAR\* Shift, VARIANT FAR\* XPos, VARIANT FAR\* YPos ) Generated when you move the mouse over the plot area.

- XPos, VARIANT FAR\* YPos )
- ◆ void **OnPlotAreaMouseUp**( short FAR\* Button, short FAR\* Shift, VARIANT FAR\* XPos, VARIANT FAR\* YPos )  
Generated when you release the mouse over the plot area.
  - ◆ void **OnPlotMouseDown**( short FAR\* Button, short FAR\* Shift, VARIANT FAR\* XData, VARIANT FAR\* YData, short FAR\* PlotIndex, long FAR\* PointIndex )  
Generated when you click the mouse on a plot.
  - ◆ void **OnPlotMouseMove**( short FAR\* Button, short FAR\* Shift, VARIANT FAR\* XData, VARIANT FAR\* YData, short FAR\* PlotIndex, long FAR\* PointIndex )  
Generated when you move the mouse over a plot.
  - ◆ void **OnPlotMouseUp**( short FAR\* Button, short FAR\* Shift, VARIANT FAR\* XData, VARIANT FAR\* YData, short FAR\* PlotIndex, long FAR\* PointIndex )  
Generated when you release the mouse over a plot.
  - ◆ void **OnReadyStateChange**()  
Generated when the ready state changes.
-

# **Knob Events**

**Function Group**



Declared in:  
**NiKnobEvents.h**





## Overview

---

The Measurement Studio knob control generates a variety of events in response to user input or changes in properties that are bound to a DataSocket source.

Complete the following steps to add an event handler for a particular event generated by the knob control.

1. Edit a dialog resource and add a Measurement Studio knob control to the dialog.
2. Right-click on the knob control and select **Events**.
3. Select the appropriate event to handle and select **Add and Edit**. By default, this creates a new member function and names it in the form shown below.

On[eventName][objectIdSuffix]

To help you find the appropriate online help, the components of the member function name are described below.

- **eventName** is the event name.
- **objectIdSuffix** is the last part of the ID assigned to the control.

For example, if you have a knob control for which the ID is IDC\_CWKNOB1 and you want to add a handler for the `click` event. The default member function name created by the ClassWizard is `OnClickCwknob1`.

4. Click **OK** to add and edit the new event handler. Notice that the prototype of the member function matches the corresponding event listed below. You can select this link to get additional information regarding the event.

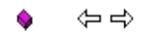
Note: Various knob events pass COM VARIANT data types as parameters to the event handler. Measurement Studio includes the `CnVariant` class, which provides a convenient interface for managing these VARIANT data types. Refer to the documentation for each event for further information.

## ◆ Functions

---

- ◆ void [OnClick\(\)](#) Generated when you click the mouse on the control.
  - ◆ void [OnCWBindingDataUpdated](#)( short Index, LPDISPATCH Data, BOOL FAR\* Ignore ) Generated when the binding data is updated.
  - ◆ void [OnCWBindingStatusUpdated](#)( short Index, long Status, long Error, LPCTSTR Message ) Generated when the status of the binding connection changes.
  - ◆ void [OnDblClick\(\)](#) Generated when you double-click the mouse on the control.
  - ◆ void [OnKeyDown](#)( short FAR\* KeyCode, short Shift ) Generated when you press a key while the control has the input focus.
  - ◆ void [OnKeyPress](#)( short FAR\* KeyAscii ) Generated when a KeyDown message generates a key while a control is active.
  - ◆ void [OnKeyUp](#)( short FAR\* KeyCode, short Shift ) Generated when you release a key while the control has the input focus.
  - ◆ void [OnMouseDown](#)( short Button, short Shift, long x, long y ) Generated when you click the mouse on the control.
  - ◆ void [OnMouseMove](#)( short Button, short Shift, long x, long y ) Generated when you move the mouse over the control.
  - ◆ void [OnMouseUp](#)( short Button, short Shift, long x, long y ) Generated when you release the mouse on the control.
  - ◆ void [OnPointerValueChanged](#)( long Pointer, VARIANT FAR\* Value ) Generated when the value of a pointer changes.
  - ◆ void [OnPointerValueCommitted](#)( long Pointer, VARIANT FAR\* Value ) Generated when the value of a pointer has stopped changing.
  - ◆ void [OnReadyStateChange](#)() Generated when the ready state changes.
-

# **N u m e r i c E d i t E v e n t s**



**Function Group**

Declared in:  
**NiNumEditEvents.h**





## Overview

---

The Measurement Studio numeric edit control generates a variety of events in response to user input or changes in properties that are bound to a DataSocket source.

Complete the following steps to add an event handler for a particular event generated by the numeric edit control.

1. Edit a dialog resource and add a Measurement Studio numeric edit control to the dialog.
2. Right-click on the numeric edit control and select **Events**.
3. Select the appropriate event to handle and select **Add and Edit**. By default, this creates a new member function for which the name is in the form shown below.

On[eventName][objectIdSuffix]

To help you find the appropriate online help, the components of the member function name are described below.

- **eventName** is the event name.
- **objectIdSuffix** is the last part of the ID assigned to the control.

For example, if you have a numeric edit control whose ID is IDC\_CWNUMEDIT1 and you want to add a handler for the `Click` event. The default member function name created by the ClassWizard is `OnClickCwnumedit1`.

4. Click **OK** to add and edit the new event handler. Notice that the prototype of the member function matches the corresponding event listed below. You can select this link to get additional information regarding the event.

Note: Various numeric edit events pass COM VARIANT data types as parameters to the event handler. Measurement Studio includes the `CNiVariant` class, which provides a convenient interface for managing these VARIANT data types. Refer to the documentation for each event for further information.

## ◆ Functions

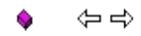
---

- ◆ void **OnClick()** Generated when you click the mouse on the control.
- ◆ void **OnCWBindingDataUpdated( short Index, LPDISPATCH Data, BOOL FAR\* Ignore )** Generated when the binding data is updated.
- ◆ void **OnCWBindingStatusUpdated( short Index, long Status, long Error, LPCTSTR Message )** Generated when the status of the binding connection changes.
- ◆ void **OnDblClick()** Generated when you double-click the mouse on the control.
- ◆ void **OnError( short Number, BSTR FAR\* Description, long Scode, LPCTSTR Source, LPCTSTR HelpFile, long HelpContext, BOOL FAR\* CancelDisplay )** Generated when the user enters an illegal value in the numeric edit control.
- ◆ void **OnIncDecButtonClicked( BOOL IncButton )** Generated when the user clicks the increment or decrement button on the numeric edit control.
- ◆ void **OnKeyDown( short FAR\* KeyCode, short Shift )** Generated when you press a key while the control has the input focus.
- ◆ void **OnKeyPress( short FAR\* KeyAscii )** Generated when a KeyDown message generates a key while a control is active.
- ◆ void **OnKeyUp( short FAR\* KeyCode, short Shift )** Generated when you release a key while the control has the input focus.
- ◆ void **OnMouseDown( short Button, short** Generated when you

- Shift, long x, long y )
- ◆ void **OnMouseMove**( short Button, short Shift, long x, long y )  
click the mouse on the control.  
Generated when you move the mouse over the control.
  - ◆ void **OnMouseUp**( short Button, short Shift, long x, long y )  
Generated when you release the mouse on the control.
  - ◆ void **OnReadyStateChange**()  
Generated when the ready state changes.
  - ◆ void **OnValueChanged**( VARIANT FAR\* NewValue, VARIANT FAR\* PreviousValue, BOOL OutOfRange )  
Generated when the value of the numeric edit control has changed.
  - ◆ void **OnValueChanging**( VARIANT FAR\* NewValue, VARIANT FAR\* AttemptedValue, VARIANT FAR\* PreviousValue, BOOL OutOfRange )  
Generated when the value of the numeric edit control is about to change.
-

# Slide Events

**Function Group**



Declared in:  
**NiSlideEvents.h**





## Overview

---

The Measurement Studio slide control generates a variety of events in response to user input or changes in properties that are bound to a DataSocket source.

Complete the following steps to add an event handler for a particular event generated by the slide control.

1. Edit a dialog resource and add a Measurement Studio slide control to the dialog.
2. Right-click on the slide control and select **Events**.
3. Select the appropriate event to handle and select **Add and Edit**. By default, this creates a new member function for which the name has the form shown below.

On[eventName][objectIdSuffix]

To help you find the appropriate online help, the components of the member function name are described below.

- **eventName** is the event name.
- **objectIdSuffix** is the last part of the ID assigned to the control.

For example, if you have a slide control for which the ID is IDC\_CWSLIDE1 and you want to add a handler for the `Click` event, the default member function name created by the ClassWizard is `OnClickCwslide1`.

4. Click **OK** to add and edit the new event handler. Notice that the prototype of the member function matches the corresponding event listed below. You can select this link to get additional information regarding the event.

Note: Various slide events pass COM VARIANT data types as parameters to the event handler. Measurement Studio includes the `CNivariant` class, which provides a convenient interface for managing these VARIANT data types. Refer to the documentation for each event for further information.

## ◆ Functions

---

- ◆ void [OnClick\(\)](#) Generated when you click the mouse on the control.
  - ◆ void [OnCWBindingDataUpdated](#)( short Index, LPDISPATCH Data, BOOL FAR\* Ignore ) Generated when the binding data is updated.
  - ◆ void [OnCWBindingStatusUpdated](#)( short Index, long Status, long Error, LPCTSTR Message ) Generated when the status of the binding connection changes.
  - ◆ void [OnDblClick\(\)](#) Generated when you double-click the mouse on the control.
  - ◆ void [OnKeyDown](#)( short FAR\* KeyCode, short Shift ) Generated when you press a key while the control has the input focus.
  - ◆ void [OnKeyPress](#)( short FAR\* KeyAscii ) Generated when a KeyDown message generates a key while a control is active.
  - ◆ void [OnKeyUp](#)( short FAR\* KeyCode, short Shift ) Generated when you release a key while the control has the input focus.
  - ◆ void [OnMouseDown](#)( short Button, short Shift, long x, long y ) Generated when you click the mouse on the control.
  - ◆ void [OnMouseMove](#)( short Button, short Shift, long x, long y ) Generated when you move the mouse over the control.
  - ◆ void [OnMouseUp](#)( short Button, short Shift, long x, long y ) Generated when you release the mouse on the control.
  - ◆ void [OnPointerValueChanged](#)( long Pointer, VARIANT FAR\* Value ) Generated when the value of a pointer changes.
  - ◆ void [OnPointerValueCommitted](#)( long Pointer, VARIANT FAR\* Value ) Generated when the value of a pointer has stopped changing.
  - ◆ void [OnReadyStateChange](#)() Generated when the ready state changes.
-