

3D Graph Component Overview

The 3D Graph component includes the functionality described below.

- You can plot three-dimensional data, including curves and surfaces. You can include three-dimensional axes and ticks on the plot.
- You can configure the control to allow your user to zoom, pan, and rotate with the mouse.
- You can configure the control to render directly to OpenGL-enabled hardware accelerator cards.
- You can use property pages to configure the control.
- You can bind the control to a DataSocket server to enable automatic read and write functionality.

Top-Level Classes

[CNiGraph3D](#) - encapsulates the interface to the Measurement Studio ActiveX 3D graph control.

Notes:

- Because the 3DGraph component includes an ActiveX control that links to the MFC DLL, projects that you design to use a Measurement Studio 3DGraph control cannot link to static MFC.
- The 3DGraph 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 3DGraph classes. To initialize COM in a thread, call the system function ::CoInitialize or the system function ::CoInitializeEx from within the thread.

3D Graph Example Programs

This topic includes summaries of and links to the example programs associated with the 3D Graph component.

Axis	The Axis example demonstrates how to set various properties of the Axis on a 3D graph.	Load example in VC++	Run example
Color Map Styles	The Color Map Styles example demonstrates how to use the color map feature of the Measurement Studio 3D Graph control. The example illustrates the available color map styles, including custom styles.	Load example in VC++	Run example
Contours	The Contours example demonstrates how to generate contours on 3D functions.	Load example in VC++	Run example
Cursors3D	The Cursors3D example demonstrates how to use the cursors feature of the Measurement Studio 3D Graph control.	Load example in VC++	Run example
Events	The Events example demonstrates the different events of the 3D graph.	Load example in VC++	Run example
Plot Styles	The Plot Styles example demonstrates how to use the the 3D graph and highlights the different plot styles of the 3D graph on different types of surfaces and curves.	Load example in VC++	Run example
Region of Interest	The Region of Interest example demonstrates how to use the mouse callbacks from the 3D graph to get mouse coordinates on the graph.	Load example in VC++	Run example
		Load	

Spinning Light	The Spinning Light example demonstrates how to change the latitude and longitude of a light source to simulate a moving light.	example in VC++	Run example
Transparency	The Transparency example demonstrates how to use the transparency property for 3-D graphs.	Load example in VC++	Run example



CNIAxes3D



Class

Declared in:
NiAxes3d.h

Overview

A `CNiAxes3D` object is a collection of axes on a 3D graph control. This collection always contains one x axis, one y axis, and one z axis.

- Use `CNiGraph3D::Axes` to obtain an axis collection for the graph.
- Use the `Item` function to access existing axes in the collection. This function can access axes by either name or index.

The default axis names are shown below.

- `XAxis`
- `YAxis`
- `ZAxis`

The associated indices are shown below.

- 1 - X Axis
- 2 - Y Axis
- 3 - Z Axis

 [Hierarchy Chart](#)

▲ Base Classes

▲ [CNiInterface](#)

◆ Data Items

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

◆ Constructors

- ◆ [**CNiAxes3D\(\)**](#) Default constructor.
- ◆ [**CNiAxes3D\(CWAxes3D_CI* pCustom, CNiInterface::ThreadAccess option \)**](#) Constructor that attaches to the specified CWAxes3D_CI pointer.
- ◆ [**CNiAxes3D\(const CNiAxes3D& source \)**](#) Copy constructor.

◆ **Destructors**

- ◆ `~CNiAxes3D()` Destructor.

◆ Functions

◆ static const IID & <u>GetIid()</u>	Returns the globally unique identifier (GUID) of the ActiveX interface to which this class connects.
◆ <u>CNiAxis3D</u>	<u>Item(long axisIndex)</u> Returns the specified axis from the collection.
◆ <u>CNiAxis3D</u>	<u>Item(const CString& axisName)</u> Returns the specified axis from the collection.
◆ const CNiAxes3D & <u>operator =(const CNiAxes3D& source)</u>	Assignment operator.

■ Examples

1. Set the caption to "Temperature" and the visibility to true on the z axis.

```
CNiGraph3D myGraph;  
CNIAxis xAxis = myGraph.Axes.Item("ZAxis");  
zAxis.Caption = "Temperature";  
zAxis.Visible = true;
```

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

```
CNiGraph3D myGraph;  
myGraph.Axes.Item(2).SetMinMax(0, 100);
```

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

```
CNiGraph3D myGraph;  
myGraph.Axes.Item("XAxis").Name = "Time";
```

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

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



Class

Declared in:
NiAxis3d.h

■ Overview

CNiAxis3D encapsulates the interface to a single axis of a 3D graph control, which allows you to modify its appearance and behavior.

Individual axes are obtained via the Axes property on a CNiGraph3D 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 hours / 60 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

◆ bool	<u>AutoScale</u>	Determines if the system automatically sets the minimum and maximum limits of the axis.
◆ <u>CString</u>	<u>Caption</u>	Specifies the text to draw on the axis.
◆ <u>CNiColor</u>	<u>CaptionColor</u>	Specifies the color used to draw the caption.
◆ <u>CNiFont</u>	<u>CaptionFont</u>	Specifies the font of the axis caption.
◆ bool	<u>CaptionNormal</u>	Draws the axis caption at the normal position when set to true.
◆ bool	<u>CaptionOpposite</u>	Draws the axis caption at the opposite position when set to true.
◆ <u>CString</u>	<u>FormatString</u>	Specifies the format string for formatting the labels on this axis.
◆ bool	<u>Inverted</u>	Specifies if the direction of an axis is inverted.
◆ <u>CNiLabels3D</u>	<u>Labels</u>	Returns a <u>CNiLabel</u> object, which specifies how labels appear on the axis.
◆ bool	<u>Log</u>	Specifies if the axis has a Log10 scale.
◆ double	<u>Maximum</u>	Specifies the maximum value of the axis.
◆ double	<u>Minimum</u>	Specifies the minimum value of the axis.
◆ <u>CString</u>	<u>Name</u>	Specifies the name of the axis.
◆ <u>CNiTicks3D</u>	<u>Ticks</u>	Returns a <u>CNiTicks</u> object, which specifies how divisions and ticks appear on this axis.
◆ <u>CNiValuePairs</u>	<u>ValuePairs</u>	Gets a collection of <u>ValuePair</u> objects associated with the control.
◆ bool	<u>Visible</u>	Specifies if the axis is visible or hidden.

◆ Constructors

◆ [**CNiAxis3D\(\)**](#) Default constructor.

◆ [**CNiAxis3D\(CWAxis3D_CI* pCustom,
CNiInterface::ThreadAccess option \)**](#) Constructor that attaches to the
specified CWAxis3D_CI pointer.

◆ [**CNiAxis3D\(const CNiAxis3D& source \)**](#) Copy constructor.

◆ **Constructors**

◆ **Destructors**

- ◆ `~CNiAxis3D()` Destructor.

◆ Functions

◆ void	<u>AutoScaleNow()</u>	Causes the axis to rescale immediately.
◆ static const IID &	<u>GetId()</u>	Returns the globally unique identifier (GUID) of the ActiveX interface to which this class connects.
◆ const CNiAxis3D & CNiAxis3D& source)	operator =(const	Assignment operator.
◆ void	<u>SetMinMax(</u> 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 z axis.
```

```
CNiGraph3D graph;  
CNiAxis3D zAxis = graph.Axes.Item("ZAxis");  
zAxis.Caption = "Temperature";  
zAxis.Visible = true;
```



▲ ◆ ♦ ♣ ♤

↔⇒

Class

Declared in:
NiContour.h

Overview

`CNiContour` encapsulates the interface to a single contour line of a 3D contour plot. This object allows you to modify the appearance and behavior of contour lines on an individual basis.

 [Hierarchy Chart](#)

▲ Base Classes

▲ [CNiInterface](#)

◆ Data Items

- ◆ [CNiColor](#) [LabelColor](#) Specifies the color of the contour label.
- ◆ [CNiFont](#) [LabelFont](#) Specifies the contour label font.
- ◆ [CString](#) [LabelFormat](#) Specifies the format string for formatting the label on this contour.
- ◆ **bool** [LabelVisible](#) Specifies if the contour line has a label.
- ◆ **double** [Level](#) Specifies the position of the contour.
- ◆ [CNiColor](#) [LineColor](#) Specifies the color of the contour line.
- ◆ LineStyles [LineStyle](#) Specifies the line style of the contour line.
- ◆ **double** [LineWidth](#) Specifies the width of the contour line in points.

◆ Constructors

◆ [CNiContour\(\)](#) Default constructor.

◆ [CNiContour\(CWContour_CI*](#)
◆ [pCustom, CNiInterface::ThreadAccess](#)
[option \)](#) Constructor that attaches to the
specified cwContour_CI pointer.

◆ [CNiContour\(const CNiContour&](#)
[source \)](#) Copy constructor.

◆ **Destructors**

- ◆ ~CNiContour() Destructor.

◆ Functions

- ◆ **static const IID & [GetIid\(\)](#)**
- ◆ **const CNiContour & [operator =\(
const
CNIContour&
source \)](#)**

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

Assignment operator.

■ Example

```
// Create a surface-contour plot.

CNiGraph3D graph;
CNiReal64Matrix data(20,20);
graph.Plot3DSimpleSurface(data);
CNiPlot3D plot = graph.Plots.Item(1);
plot.Style = CNiPlot3D::SurfaceContour;

// Change the label font on the first contour line in the plot to 12
// Courier and the contour line style to dashed.

CNiContour contour = plot.Contours.Item(1);
contour.LabelFont.Name = "Courier";
contour.LabelFont.Size = 12;
contour.LineStyle = CNiContour::Dash;
```

 CNiContours      

Class

Declared in:
NiContours.h

Overview

`CNiContours` encapsulates the interface to the `Contours` property of a `CNiPlot3D` object. This object allows you to modify the appearance and behavior of all contour lines in a 3D contour plot.

 [Hierarchy Chart](#)

▲ Base Classes

▲ [CNiInterface](#)

◆ Data Items

◆ double	Anchor	Specifies the value of the contour anchor.
◆ bool	AnchorEnabled	Enables the contour anchor if true.
◆ Bases	Basis	Specifies the basis of the plot contours.
◆ short	Count	The number of objects in the collection.
◆ double	Interval	Specifies the distance between each contour level.
◆ CNIColor	LabelColor	Sets the label color for all contours in the collection.
◆ CNIFont	LabelFont	Sets the label font for all contours in the collection.
◆ CString	LabelFormat	Sets the label format string for all contours in the collection.
◆ bool	LabelVisible	Sets the label visibility for all contours in the collection.
◆ long	Levels	Specifies the number of contour levels for the plot.
◆ CNIColor	LineColor	Sets the line color for all contours in the collection.
◆ LineStyles	LineStyle	Sets the line style for all contours in the collection.
◆ double	LineWidth	Sets the line width for all contours in the collection.

◆ Constructors

◆ [CNiContours\(\)](#) Default constructor.

◆ [CNiContours\(CWContours_CI*](#)
◆ [pCustom, CNiInterface::ThreadAccess](#)
[option \)](#) Constructor that attaches to the
specified `CWContours_CI`
pointer.

◆ [CNiContours\(const CNiContours&](#)
[source \)](#) Copy constructor.

◆ **Destructors**

- ◆ `~CNiContours()` Destructor.

◆ Functions

◆ void	Add(double Level)	Adds a new contour level to the collection of contours.
◆ static const IID &	GetIid()	Returns the globally unique identifier (GUID) of the ActiveX interface to which this class connects.
◆ void	GetLevelList(CNiReal64Vector& list)	Gets the contour levels of the plot.
◆ <u>CNiContour</u>	Item(long itemIndex)	Returns the specified contour object from the current collection of contours.
◆ const CNiContours &	operator =(const source)	Assignment operator.
◆ void	Remove(long itemIndex)	Removes the specified contour from the current collection of contours.
◆ void	RemoveAll()	Removes all contours from the plot.
◆ void	SetLevelList(const CNiReal64Vector& list)	Sets the contour levels for the plot.

■ Example

```
// Create a surface-contour plot.

CNiGraph3D graph;
CNiReal64Matrix data(20,20);
graph.Plot3DSimpleSurface(data);
CNiPlot3D plot = graph.Plots.Item(1);
plot.Style = CNiPlot3D::SurfaceContour;

// Change the label font on all contour lines in the plot to 12pt
// Courier and the contour line style to dashed.

CNiContours contours = plot.Contours;
contours.LabelFont.Name = "Courier";
contours.LabelFont.Size = 12;
contours.LineStyle = CNiContour::Dash;
```



Class

Declared in:
NiCursor3D.h

Overview

`CNiCursor3D` encapsulates the interface to a single cursor on a graph control, which allows you to modify its appearance and behavior. You use cursors to enable your users to identify specific points on your 3D graph plots.

You get individual cursors using the `Cursors` property on a `CNiGraph3D` object.

 [Hierarchy Chart](#)

▲ Base Classes

▲ [CNiInterface](#)

◆ Data Items

◆ long	Column	Specifies the row index of the point associated with the cursor on the plot.
◆ bool	Enabled	Specifies if the cursor generates mouse events or if you can drag the cursor in cursor tracking mode.
◆ CNiFont	Font	Specifies the cursor label font.
◆ CNiColor	LineColor	Specifies the color of the cursor point.
◆ LineStyles	LineStyle	Specifies the style of the cursor lines.
◆ double	LineWidth	Specifies the width of the cursor line.
◆ CString	Name	Specifies the name of the cursor.
◆ bool	NameVisible	Specifies if the cursor name is displayed next to the cursor.
◆ CNiColor	PlaneColor	Specifies the color of the cursor point.
◆ CNiPlot3D	Plot	Specifies the plot associated with the cursor.
◆ CNiColor	PointColor	Specifies the color of the cursor point.
◆ double	PointSize	Specifies the size of the cursor point in points.
◆ PointStyles	PointStyle	Specifies the style of the cursor point.
◆ bool	PositionVisible	Specifies if the cursor position is displayed next to the cursor.
◆ long	Row	Specifies the row index of the point associated with the cursor on the plot.

◆ SnapModes	SnapMode	Specifies the snap mode of the cursor.
◆ CNIColor	TextBackColor	Specifies the color of the cursor point.
◆ long	TextBackgroundTransparency	Specifies the transparency of the background drawn behind the cursor name and position text.
◆ CNIColor	TextColor	Specifies the color of the cursor point.
◆ long	Transparency	Indicates the percentage of transparency of the cursor planes.
◆ bool	Visible	Specifies if the cursor is visible or hidden.
◆ double	XPosition	Current x axis position of the cursor.
◆ bool	XYPlaneVisible	Specifies if a plane is displayed at the cursor Z value.
◆ bool	XZPlaneVisible	Specifies if a plane is displayed at the cursor Y value.
◆ double	YPosition	Current y axis position of the cursor.
◆ bool	YZPlaneVisible	Specifies if a plane is displayed at the cursor X value.
◆ double	ZPosition	Current z axis position of the cursor.

◆ Constructors

◆ [CNiCursor3D\(\)](#) Default constructor.

◆ [CNiCursor3D\(CWCursor3D_CI* pCustom, CNiInterface::ThreadAccess option \)](#) Constructor that attaches to the specified `CWCursor3D_CI` pointer.

◆ [CNiCursor3D\(const CNiCursor3D& source \)](#) Copy constructor.

◆ **Destructors**

- ◆ `~CNiCursor3D()` Destructor.

◆ Functions

◆ static const IID &	<u>GetId()</u>	Returns the globally unique identifier (GUID) of the ActiveX interface to which this class connects.
◆ const CNiCursor3D &	CNiCursor3D& source)	operator =(const <u>SetColor(const</u> <u>CNiColor & color)</u>
◆ void	<u>SetColor(const</u> <u>CNiColor & color)</u>	Assignment operator. Sets the all the colors of the cursor at the same time.
◆ void	<u>SetPosition(double</u> xPosition, double yPosition, double zPosition)	<u>SetPosition(double</u> xPosition, double yPosition, double zPosition)

■ Example

```
// Change the snap mode of the first cursor on the graph.  
CNiGraph3D graph;  
CNiCursor3D cursor = graph.Cursors.Item(1);  
cursor.SnapMode = CNiCursor3D::SnapNearestPlot;
```



CNiCursors3D ▲ ◆ ◇ ◆ ◆ ☐ ⇔ ⇕

Class

Declared in:
NiCursors3D.h

Overview

`CNiCursors3D` encapsulates the interface to the `Cursors` property of a `CNiGraph3D` object, which allows you to access and remove cursors associated with the graph control. You use cursors to enable your users to identify specific points on your 3D graph plots.

 [Hierarchy Chart](#)

▲ Base Classes

▲ [CNiInterface](#)

◆ Data Items

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

◆ Constructors

◆ [CNiCursors3D\(\)](#) Default constructor.

◆ [CNiCursors3D\(CW Cursors3D_CI*](#)
◆ [pCustom, CNiInterface::ThreadAccess](#)
[option \)](#) Constructor that attaches to the
specified CW Cursors3D_CI
pointer.

◆ [CNiCursors3D\(const CNiCursors3D&](#)
[source \)](#) Copy constructor.

◆ **Destructors**

- ◆ `~CNiCursors3D()` Destructor.

◆ Functions

◆ [CNiCursor3D](#)

[Add\(\)](#)

Adds a cursor to the collection and returns the new cursor.

◆ **static const IID &**

[GetId\(\)](#)

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

◆ [CNiCursor3D](#)

[Item\(long cursorIndex \)](#)

Returns the specified cursor from the collection.

◆ [CNiCursor3D](#)

[Item\(const cString& cursorName \)](#)

Returns the specified cursor from the collection.

◆ **const CNiCursors3D &**

**operator =(
const
CNiCursors3D&
source)**

Assignment operator.

◆ **void**

[Remove\(long cursorIndex \)](#)

Removes the specified cursor from the collection.

◆ **void**

[Remove\(const cString& cursorName \)](#)

Removes the specified cursor from the collection.

◆ **void**

[RemoveAll\(\)](#)

Removes all cursors from the collection.

■ Examples

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

```
CNiGraph graph;
CNiCursor3D cursor = graph.Cursors.Item(1);
cursor.SnapMode = CNiCursor3D::SnapNearestPlot;
cursor.Visible = true;
```

2. Set the point style.

```
CNiGraph graph;
graph.Cursors.Item(1).PointStyle = CNiCursor3D::WireframeCube;
```

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

```
CNiGraph graph;
CNiCursor3D 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;
```



▲ ♦ ♦ ♦ ♦

↔

Class

Declared in:
NiGraph3d.h

■ Overview

CNiGraph3D encapsulates the interface to the Measurement Studio ActiveX 3D graph control, which allows you to plot three-dimensional data.

CNiGraph3D 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 [Graph3D Events](#) page.

Features

- Multiple plot styles: point-line, line-point, hidden-line, contour, surface, surface-line, surface-contour, and surface-normal.
- Multiple plots with individual properties such as name, line and point style, width, and base value.
- Configurable axes, including customizable ticks, labels, value pairs, and captions.
- Legends
- Cartesian, cylindrical, and spherical coordinate systems.
- Color maps.
- Transparency
- Plane projections.
- Orthographic and perspective viewing.
- Lighting
- Built-in format styles for labels including scientific, symbolic engineering, scaling, time, and date.
- Rotation, 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: 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 hours / 60 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.

[!\[\]\(b66725e52a8c81be973cd518958addb9_img.jpg\) Hierarchy Chart](#)

▲ Base Classes

▲ CNiControl

◆ Data Items

◆ CNiColor	AmbientLightColor	Specifies the ambient light color for the graph when lighting is enabled.
◆ CNiAxes3D	Axes	Gets a collection of axis objects associated with the control.
◆ CNiColor	BackColor	Specifies the color for the background of the caption.
◆ CString	Caption	Specifies the caption to be drawn on the graph.
◆ CNiColor	CaptionColor	Specifies the color of the caption.
◆ bool	ClipData	Specifies if the function should clip the displayed data.
◆ CNiCursors3D	Cursors	Gets a collection of 3D cursor objects associated with the control.
◆ bool	Dither	Specifies the dither mode of the graph.
◆ bool	Enabled	Specifies if the graph generates any events.
◆ bool	FastDraw	Specifies how the system draws the graph during interactive operations such as rotating, zooming, and panning.
◆ CNiFont	Font	Specifies the font for the caption.
◆ CNiColor	GraphFrameColor	Specifies the color for the graph frame.
◆ bool	GraphFrameVisible	Specifies if the graph frame is visible.

◆ CNiColor	GridFrameColor	Specifies the color of the grid frame.
◆ bool	GridSmoothing	Specifies the smoothing mode for grid lines.
◆ bool	GridXY	Draws the X-Y grid plane when true.
◆ bool	GridXZ	Draws the X-Z grid plane when true.
◆ bool	GridYZ	Draws the Y-Z grid plane when true.
◆ bool	ImmediateUpdates	Specifies if the graph draws new data as soon as it is available, or if the form refreshes the graph when it draws other controls.
◆ KeyboardModes	KeyboardMode	Specifies how the control handles keyboard input from the user.
◆ bool	Lighting	Enables graph lighting when true.
◆ CNiLights	Lights	Gets a collection of light objects associated with the control.
◆ CNiColor	PlotAreaColor	Specifies the background color of the plot area.
◆ CNiPlots3D	Plots	Gets a collection of 3D plot objects associated with the control.
◆ CNiPlot3D	PlotTemplate	Returns the 3D plot object to use as a template for new plots.
◆ ProjectionStyles	ProjectionStyle	Specifies the projection style of the graph.
◆ long	ReadyState	Returns the ready state.
		Determines the type of events generated and other

◆ TrackModes	TrackMode	automatic processing (for example, how the mouse interacts with the graph during run time).
◆ bool	Use3DHardwareAcceleration	Determines if the 3D graph control renders directly to the display adapter.
◆ bool	ViewAutoDistance	Specifies how the viewing distance is set.
◆ double	ViewDistance	Specifies the distance of the viewing position from the center of the view.
◆ double	ViewLatitude	Specifies the latitude of the viewing position.
◆ double	ViewLongitude	Specifies the longitude of the viewing position.
◆ ViewModes	ViewMode	Specifies the viewing position of the graph.
◆ double	ViewXCenter	Specify the X coordinates of the center of the view.
◆ double	ViewYCenter	Specify the Y coordinates of the center of the view.
◆ double	ViewZCenter	Specify the Z coordinates of the center of the view.
◆ bool	Windowless	Specifies if the control has a window.

◆ Constructors

◆ [CNiGraph3D\(CNiInterface::ThreadAccess option =
CNiInterface::MultipleThreadsWithCaching \)](#)

◆ **Destructors**

◆ `~CNiGraph3D()`

◆ Functions

◆ void	<u>AboutBox()</u>	Displays the About Box for the control.
◆ void	<u>ClearData()</u>	Clears data in all plots.
◆ CNiPicture	<u>ControlImage()</u>	Returns an image of the entire control.
◆ virtual BOOL	<u>Create(LPCTSTR lpszClassName, LPCTSTR lpszWindowName, DWORD dwStyle, const RECT& rect, CWnd* pParentWnd, UINT nID, CCreateContext* pContext = NULL)</u>	Creates the ActiveX control that is represented in the MFC program by this object.
◆ BOOL	<u>Create(LPCTSTR lpszWindowName, DWORD dwStyle, const RECT& rect, CWnd* pParentWnd, UINT nID, CFile* pPersist = NULL, BOOL bStorage = FALSE, BSTR bstrLicKey = NULL)</u>	Creates the ActiveX control that is represented in the MFC program by this object.
◆ BOOL	<u>CreateControl(LPCTSTR lpszClass, LPCTSTR lpszWindowName, DWORD dwStyle, const RECT& rect, CWnd* pParentWnd, UINT nID, CFile* pPersist = NULL, BOOL bStorage = FALSE, BSTR bstrLicKey = NULL)</u>	Creates the ActiveX control that is represented in the MFC program by this object.

◆ BOOL	CreateControl(REFCLSID clsid, LPCTSTR lpszWindowName, DWORD dwStyle, const RECT& rect, CWnd* 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	ExportStyle(LPCTSTR FileName)	Exports the style of the control to a file.
◆ static const CLSID &	GetClid()	Returns the globally unique identifier (GUID) of the ActiveX control to which this class connects.
◆ static const IID &	GetId()	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	Plot3DCurve(const CNiVector& x, const CNiVector& y, const CNiVector& z, const CNiVector& w)	Plots a set of vectors as a parametric curve plot.

◆ void	Plot3DCurve(const CNiVector& x, const CNiVector& y, const CNiVector& z)	Plots a set of vectors as a parametric curve plot.
◆ void	Plot3DMesh(const CNiVector& x, const CNiVector& y, const CNiVector& z, const CNiVector& w)	Plots three (or four) 1D arrays of data as a triangulated surface.
◆ void	Plot3DMesh(const CNiVector& x, const CNiVector& y, const CNiVector& z)	Plots three (or four) 1D arrays of data as a triangulated surface.
◆ void	Plot3DParametricSurface(const CNiMatrix& X, const CNiMatrix& Y, const CNiMatrix& Z, const CNiMatrix& W)	Plots a set of matrices as a parametric surface plot.
◆ void	Plot3DParametricSurface(const CNiMatrix& X, const CNiMatrix& Y, const CNiMatrix& Z)	Plots a set of matrices as a parametric surface plot.
◆ void	Plot3DSimpleSurface(const CNiMatrix& Z, const CNiMatrix& W)	Plots a matrix of data as a simple surface plot.
◆ void	Plot3DSimpleSurface(const CNiMatrix& Z)	Plots a matrix of data as a simple surface plot.
	Plot3DSurface(const CNiVector& x, const CNiVector& y, const	Plots a matrix

◆ void	<u>CNiMatrix& Z, const CNiMatrix& W)</u>	of data as a surface plot.
◆ void	<u>Plot3DSurface(const CNiVector& x, const CNiVector& y, const CNiMatrix& Z)</u>	Plots a matrix of data as a surface plot.
◆ void	<u>SetDefaultView()</u>	Restores the default viewing parameters.
◆ void	<u>ValidateControl()</u>	Validates the current state of the control.



↔⇒

Class

Declared in:
NiLabels3d.h

Overview

`CNiLabels3D` encapsulates the interface to the `Labels` property of the `CNiAxis3D` object. You use 3D labels to annotate the axes on 3D graph controls.

 [Hierarchy Chart](#)

▲ Base Classes

▲ [CNiInterface](#)

◆ Data Items

- ◆ [CNiColor](#) **Color** Specifies the color of the labels.
- ◆ [CNiFont](#) **Font** Specifies the font of the labels.
- ◆ **bool** [Normal](#) Specifies if labels appear at the normal axis position.
- ◆ **bool** [Opposite](#) Specifies if labels appear at the opposite axis position.

◆ Constructors

◆ [CNiLabels3D\(\)](#) Default constructor.

◆ [CNiLabels3D\(CWLabels3D_CI* pCustom, CNiInterface::ThreadAccess option \)](#) Constructor that attaches to the specified CWLabels3D_CI pointer.

◆ [CNiLabels3D\(const CNiLabels3D& source \)](#) Copy constructor.

◆ **Destructors**

- ◆ `~CNiLabels3D()` Destructor.

◆ Functions

- ◆ **static const IID &** [**GetId\(\)**](#) Returns the globally unique identifier (GUID) of the ActiveX interface to which this class connects.

 - ◆ **const CNiLabels3D &** **operator = (**
 const
 CNiLabels3D&
 source) Assignment operator.
-



Class

Declared in:
NiLight.h

Overview

`CNiLight` encapsulates the interface to a single light on a 3D graph object, which allows you to modify its appearance and behavior.

Lights are positioned within the world coordinates of the graph using the following properties:

- Longitude
- Latitude
- Distance

You get individual lights using the `Lights` property on a `CNiGraph3D` 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 hours / 60 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

◆ LightAttenuationStyles	Attenuation	Specifies the attenuation style of the light source.
◆ CNiColor	Color	Specifies the color of the light source.
◆ double	Distance	Specifies the distance of the light source from the origin of the graph.
◆ bool	Enabled	Enables lighting when set to true.
◆ double	Latitude	Specifies the latitudinal position of the light source.
◆ double	Longitude	Specifies the longitudinal position of the light source.

◆ Constructors

◆ [CNiLight\(\)](#) Default constructor.

◆ [CNiLight\(CWLight_CI* pCustom,](#) [CNiInterface::ThreadAccess option \)](#) Constructor that attaches to the specified `CWLIGHT_CI` pointer.

◆ [CNiLight\(const CNiLight& source \)](#) Copy constructor.

◆ **Destructors**

- ◆ ~CNiLight() Destructor.

◆ Functions

- ◆ **static const IID & [GetIid\(\)](#)** Returns the globally unique identifier (GUID) of the ActiveX interface to which this class connects.

- ◆ **const CNiLight & [operator =\(
 const
 CNIlight&
 source \)](#)** Assignment operator.

■ Example

```
// Set the attenuation mode of the third light in a 3D graph.  
CNiGraph3D graph;  
CNiLight light = graph.Lights.Item(3);  
light.Attenuation = CNiLight::Quadratic;
```



▲ ◆ ♦ ♣ ♤ ☰ ⇔ ⇕

Class

Declared in:
NiLights.h

■ Overview

`CNiLights` encapsulates the `Lights` property of a `CNiGraph3D` object. It provides access to the individual lights.

Use the `Item` function to get a particular light object and the `Count` property to retrieve the number of lights in the graph.

Note: Currently, there are always four lights associated with a 3D graph.

[!\[\]\(dfb61f5a70f03ec64aa6bce7cbd06811_img.jpg\) Hierarchy Chart](#)

▲ Base Classes

▲ [CNiInterface](#)

◆ Data Items

- ◆ **short** [Count](#) Returns the number of lights in the collection.

◆ Constructors

◆ [CNiLights\(\)](#) Default constructor.

◆ [CNiLights\(CWLights_CI* pCustom,](#) [CNiInterface::ThreadAccess option \)](#) Constructor that attaches to the specified `CWLights_CI` pointer.

◆ [CNiLights\(const CNiLights& source \)](#) Copy constructor.

◆ **Destructors**

- ◆ ~CNiLights() Destructor.

◆ Functions

◆ static const IID & GetIid()	Returns the globally unique identifier (GUID) of the ActiveX interface to which this class connects.
◆ CNiLight	Item(long lightIndex) Returns the specified light from the current collection of lights.
◆ const CNiLights & operator =(const CNiLights& source)	Assignment operator.



CNiPlot3D

Class

Declared in:
NiPlot3d.h

■ Overview

CNiPlot3D encapsulates the interface to a single plot of 3D graph control, which allows you to modify its appearance and behavior.

You obtain individual plots using the `Plots` property on a CNiGraph3D 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 hours / 60 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

◆ bool	AutoSize	Specifies if the extents of the data in the plot affect the extents of an autoscaling axis.
◆ bool	CacheData	Improves drawing performance by caching intermediate calculations when set to true.
◆ bool	ColorMapAutoScale	Specifies if the color map rescales its values when the plot data changes.
◆ CNiColorVector	ColorMapColors	ColorMapColors and ColorMapValues specify a custom color map.
◆ bool	ColorMapInterpolate	Specifies if the color map interpolates between colors.
◆ bool	ColorMapLog	Specifies if the color map interpolates logarithmically.
◆ ColorMapStyles	ColorMapStyle	Specifies the color map used by the plot.
◆ CNiReal64Vector	ColorMapValues	ColorMapColors and ColorMapValues specify a custom color map.
◆ CNiContours	Contours	Gets a collection of contour objects associated with the plot.
◆ CoordinateSystems	CoordinateSystem	Specifies the coordinate system of the plot.
◆ bool	Enabled	Specifies if the plot generates mouse events when CNiGraph3D.TrackMode = TrackAllEvents and the plot is visible.
◆ CNiColor	FillColor	Specifies the color to use for drawing the plot surface.
FillStyles	FillStyle	

◆		Specifies the fill style of the plot.
◆ CNiColor	LineColor	Specifies the color of lines for connecting points in the plot.
◆ LineStyles	LineStyle	Specifies the style of lines for connecting points on a plot.
◆ double	LineWidth	Specifies the width of the plotting line.
◆ bool	MultiPlot	Determines if the <code>CNiGraph3D</code> plot functions can use this plot.
◆ CString	Name	Specifies the name of the plot.
◆ CNiColor	PointColor	Specifies the color for points on a plot.
◆ long	PointFrequency	Specifies the frequency of points in the plot.
◆ double	PointSize	Specifies the size of the plot points in points.
◆ PointStyles	PointStyle	Specifies the style of the points in the plot.
◆ bool	ProjectionXY	ProjectionXY draws the X-Y plane projection of the plot when true.
◆ bool	ProjectionXZ	ProjectionXZ draws the X-Z plane projection of the plot when true.
◆ bool	ProjectionYZ	ProjectionYZ draws the Y-Z plane projection of the plot when true.
◆ bool	ShowProjectionsOnly	Defines the show-projections-only mode.
◆ PlotStyles	Style	Specifies the style of the plot.
◆ long	Transparency	Indicates the percentage of transparency.
◆ bool	Visible	Specifies if the plot is visible or hidden.
◆ CNiAxis3D	XAxis	Gets the x axis for the plot.

◆ [CNiAxis3D](#)

[YAxis](#)

Gets the y axis for the plot.

◆ [CNiAxis3D](#)

[ZAxis](#)

Gets the z axis for the plot.

◆ Constructors

◆ [CNiPlot3D\(\)](#) Default constructor.

◆ [CNiPlot3D\(CWPlot3D_CI* pCustom,
CNiInterface::ThreadAccess option \)](#) Constructor that attaches to the
specified CWPlot3D_CI pointer.

◆ [CNiPlot3D\(const CNiPlot3D& source \)](#) Copy constructor.

◆ **Destructors**

- ◆ `~CNiPlot3D()` Destructor.

◆ Functions

◆ void	<u>ClearData()</u>	Clears the data currently associated with the plot.
◆ static const IID &	<u>GetIid()</u>	Returns the globally unique identifier (GUID) of the ActiveX interface to which this class connects.
◆ const CNiPlot3D &	operator = (const CNiPlot3D& source)	Assignment operator.
◆ void	<u>Plot3DCurve(const CNiVector& x, const CNiVector& y, const CNiVector& z, const CNiVector& w)</u>	Plots a set of vectors as a parametric curve plot.
◆ void	<u>Plot3DCurve(const CNiVector& x, const CNiVector& y, const CNiVector& z)</u>	Plots a set of vectors as a parametric curve plot.
◆ void	<u>Plot3DMesh(const CNiVector& x, const CNiVector& y, const CNiVector& z, const CNiVector& w)</u>	Plots three (or four) 1D arrays of data as a triangulated surface.
	<u>Plot3DMesh(const CNiVector& x, const</u>	Plots three (or four) 1D arrays of data as a

◆ void	<u>Plot3DParametricSurface(const CNiVector& y, const CNiVector& z)</u>	triangulated surface.
◆ void	<u>Plot3DParametricSurface(const CNiMatrix& X, const CNiMatrix& Y, const CNiMatrix& Z, const CNiMatrix& W)</u>	Plots a set of matrices as a parametric surface plot.
◆ void	<u>Plot3DParametricSurface(const CNiMatrix& X, const CNiMatrix& Y, const CNiMatrix& Z)</u>	Plots a set of matrices as a parametric surface plot.
◆ void	<u>Plot3DSimpleSurface(const CNiMatrix& Z, const CNiMatrix& W)</u>	Plots a matrix of data as a simple surface plot.
◆ void	<u>Plot3DSimpleSurface(const CNiMatrix& Z)</u>	Plots a matrix of data as a simple surface plot.
◆ void	<u>Plot3DSurface(const CNiVector& x, const CNiVector& y, const CNiMatrix& Z, const CNiMatrix& W)</u>	Plots a matrix of data as a surface plot.
◆ void	<u>Plot3DSurface(const CNiVector& x, const CNiVector& y, const CNiMatrix& Z)</u>	Plots a matrix of data as a surface plot.

■ Example

```
// Add a surface plot to the graph.  
  
CNiGraph3D graph;  
CNiReal64Matrix surface(20, 20);  
graph.SimpleSurfacePlot(surface);  
  
// Get the plot from the graph and change its color map style.  
  
CNiPlot3D plot = graph.Plots.Item(1);  
plot.ColorMapStyle = CNiPlot3D::ColorSpectrum;
```



▲ ◆ ♦ ◆ ♦ ◆

↔ ↔

Class

Declared in:
NiPlots3d.h

■ Overview

`CNiPlots3D` encapsulates the interface to the `Plots` property of a `CNiGraph3D` object, which allows you to access and remove plots associated with the 3D graph control.

- Use the `CNiGraph3D::Plots` property to obtain the plots collection for the graph.
- Use the `Add` function to create additional plots. `Add` returns a `CNiPlot3D` 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

- ◆ [**CNiPlots3D\(\)**](#) Default constructor.
- ◆ [**CNiPlots3D\(CWPlots3D_CI* pCustom, CNiInterface::ThreadAccess option \)**](#) Constructor that attaches to the specified CWPlots3D_CI pointer.
- ◆ [**CNiPlots3D\(const CNiPlots3D& source \)**](#) Copy constructor.

◆ **Destructors**

- ◆ `~CNiPlots3D()` Destructor.

◆ Functions

◆ CNiPlot3D	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.
◆ CNiPlot3D	Item(long plotIndex)	Returns the specified plot from the collection.
◆ CNiPlot3D	Item(const CString& plotName)	Returns the specified plot from the collection.
◆ const CNiPlots3D &	operator =(const CNiPlots3D& 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;  
CNiPlot3D 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 = CNiPlot3D::PointAsterisk;
```

3. Add a new plot to the graph.

```
CNiGraph graph;  
CNiPlot3D 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;
```



↔

Class

Declared in:
NiTicks3d.h

■ Overview

`CNiTicks3D` encapsulates the interface to the `Ticks` property of a `CNiAxis3D` object, which allows you to modify the appearance and behavior of the tick marks on a 3D 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 hours / 60 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

◆ bool	AutoDivisions	Specifies if divisions are automatically calculated.
◆ bool	Inside	Specifies if ticks are draw on the inside of the axis plane.
◆ double	MajorDivisions	Specifies the number of major divisions.
◆ bool	MajorGrid	Specifies if major grid lines are drawn.
◆ CNIColor	MajorGridColor	Specifies the color of major grid lines.
◆ CNIColor	MajorTickColor	Specifies the color of major grid ticks.
◆ bool	MajorTicks	Specifies if major grid ticks are drawn.
◆ double	MajorUnitsBase	Specifies the base number for calculating ticks.
◆ double	MajorUnitsInterval	Specifies the number of units between major divisions.
◆ double	MinorDivisions	Specifies the number of minor divisions for each major division.
◆ bool	MinorGrid	Specifies if minor grid lines are drawn.
◆ CNIColor	MinorGridColor	Specifies the color of minor grid lines.
◆ CNIColor	MinorTickColor	Specifies the color of minor grid ticks.
◆ bool	MinorTicks	Specifies if minor grid ticks are drawn.
◆ double	MinorUnitsInterval	Specifies the number of units between minor divisions.
◆ bool	Normal	Specifies if tick marks are drawn at the normal axis position.
◆ bool	Opposite	Specifies if tick marks are drawn at the opposite axis position.
◆ bool	Outside	Specifies if ticks are drawn on the outside of the axis plane.

◆ Constructors

◆ [CNiTicks3D\(\)](#) Default constructor.

◆ [CNiTicks3D\(CWTicks3D_CI*](#)
◆ [pCustom, CNiInterface::ThreadAccess](#)
[option \)](#) Constructor that attaches to the
specified CWTicks3D_CI pointer.

◆ [CNiTicks3D\(const CNiTicks3D&](#)
[source \)](#) Copy constructor.

◆ **Constructors**

- ◆ `~CNiTicks3D()` Destructor.

◆ Functions

- ◆ **static const IID & [GetId\(\)](#)** Returns the globally unique identifier (GUID) of the ActiveX interface to which this class connects.
- ◆ **const CNiTicks3D & [operator=\(
const
CNITicks3D&
source \)](#)** Assignment operator.

■ Example

```
// Changes the number of major divisions on the x axis of a 3D graph
// control.

CNiGraph3D graph;
CNiTicks ticks = graph.Axes.Item("XAxis").Ticks;
ticks.MajorDivisions = 20;
```



Graph3D Events ♦ ⇔

Function Group

Declared in:
NiGraph3DEvents.h

Overview

The Measurement Studio 3D 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 3D graph control.

1. Edit a dialog resource and add a Measurement Studio 3D graph control to the dialog.
2. Right-click on the 3D 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 3D graph control for which the ID is `IDC_CWGRAPH3D1` and you want to add a handler for the `Click` event. The default member function name created by the ClassWizard is `OnClickCwgraph3d1`.

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 3D 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 [OnClick\(\)](#)

Generated when you click the mouse on the control.

◆ void [OnCursorChange\(long FAR* CursorIndex, double FAR* XPosition, double FAR* YPosition, double FAR* ZPosition, BOOL FAR* Tracking \)](#)

Generated when you reposition a cursor with the mouse.

◆ 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.

[OnMouseMove\(short Button, short Shift, long X, long Y \)](#)

Generated when you move the

- ◆ **void long Y)** mouse over the control.

- ◆ **void OnMouseUp(short Button, short Shift, long X, long Y)** Generated when you release the mouse on the control.

- ◆ **void OnPan(VARIANT FAR* NewXCenter, VARIANT FAR* NewYCenter, VARIANT FAR* NewZCenter)** Generated when you pan the graph up and down or left and right.

- ◆ **void OnPlotAreaMouseDown(short FAR* Button, short FAR* Shift, VARIANT FAR* XNear, VARIANT FAR* YNear, VARIANT FAR* ZNear, VARIANT FAR* XFar, VARIANT FAR* YFar, VARIANT FAR* ZFar)** Generated when you click the mouse on the plot area.

- ◆ **void OnPlotAreaMouseMove(short FAR* Button, short FAR* Shift, VARIANT FAR* XNear, VARIANT FAR* YNear, VARIANT FAR* ZNear, VARIANT FAR* XFar, VARIANT FAR* YFar, VARIANT FAR* ZFar)** Generated when you move the mouse over the plot area.

- ◆ **void OnPlotAreaMouseUp(short FAR* Button, short FAR* Shift, VARIANT FAR* XNear, VARIANT FAR* YNear, VARIANT FAR* ZNear, VARIANT FAR* XFar, VARIANT FAR* YFar, VARIANT FAR* ZFar)** Generated when you release the mouse over the plot area.

- ◆ **void OnPlotMouseDown(short FAR* Button, short FAR* Shift, VARIANT FAR* XData, VARIANT FAR* YData, VARIANT FAR* ZData, short FAR* PlotIndex, long FAR* PointI, long FAR* PointJ)** Generated when you click the mouse on a plot.

- ◆ **[OnPlotMouseMove](#)**(**short FAR*** Button, **short FAR*** Shift, VARIANT FAR* XData, VARIANT FAR* YData, VARIANT FAR* ZData, **short FAR*** PlotIndex, **long FAR*** PointI, **long FAR*** PointJ)
Generated when you move the mouse over a plot.

 - ◆ **[OnPlotMouseUp](#)**(**short FAR*** Button, **short FAR*** Shift, VARIANT FAR* XData, VARIANT FAR* YData, VARIANT FAR* ZData, **short FAR*** PlotIndex, **long FAR*** PointI, **long FAR*** PointJ)
Generated when you release the mouse over a plot.

 - ◆ **[OnReadyStateChange](#)**()
Generated when the ready state changes.

 - ◆ **[OnRotate](#)**(VARIANT FAR* NewLatitude, VARIANT FAR* NewLongitude)
Generated when you rotate the graph.

 - ◆ **[OnZoom](#)**(VARIANT FAR* NewDistance)
Generated when you zoom in or out on the plot.
-