

GLG Toolkit FAQs : AEROPGLG-FAQ2

GLG Toolkit FAQs

GLG Widgets FAQs:

- [General Widget Questions](#)

- [Chart and Graph Widgets FAQ](#)

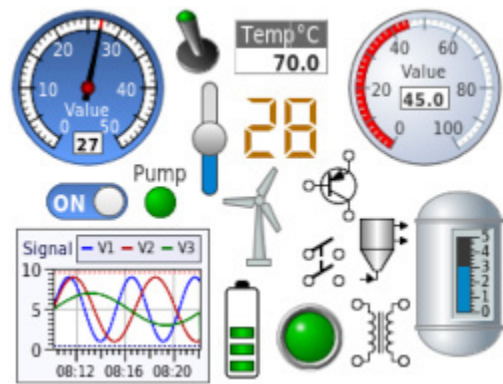
- [Dials and Meters FAQ](#)

[Online Demos](#)

[Widget Catalog:](#)

- [Charts and Graphs](#)
- [Dials and Meters](#)
- [Avionics Gauges](#)

[Documentation and Tutorials](#)



[General Widget Questions](#)

[What is a GLG Widget?](#)

[Where are widget drawings located in the GLG directory?](#)

[What do I do with the widget's .g file?](#)

[How do I use a widget in my application?](#)

[Are there programming examples of using GLG Widgets?](#)

[How do I load a widget in the Graphics Builder?](#)

[How do I modify a widget?](#)

[How do I browse widget's resources?](#)

[Can I edit objects inside the widget's drawing directly?](#)

[Can I add new objects to a widget or delete the ones I do not need?](#)

[How do I animate a widget in the Builder?](#)

[How do I supply data to animate a widget at run time?](#)

[How do I create a new widget from scratch?](#)

[I need to display several widgets. Do I need a separate Bean / Control container for each widget?](#)

[How do I create a dashboard containing several widgets?](#)

[Dials and Meters FAQ](#)

[What is a GLG dial widget?](#)

[How do I edit a dial widget's properties?](#)

[How do I prototype a dial widget in the Graphics Builder?](#)

[How do I use a dial widget in my application?](#)

[How do I handle the dial widget's input?](#)

[I want to use a dial widget for output only. How do I disable the dial's input?](#)

[How do I add more than one needle to a dial?](#)

[Chart and Graph Widgets, General Questions](#)

[How do I use a chart or a graph widget in my application?](#)

[How do I prototype a chart or a graph in the Graphics Builder?](#)

[Real-Time Charts](#)

[What is the difference between the Real-Time Charts and the Graph Widgets](#)

[How do I edit properties of the Real-Time Chart widget](#)

[How do I supply data to a Real-Time Chart](#)

[2D and 3D Graphs](#)

[What is a GLG graph widget?](#)

[How do I modify graph's resources and geometry?](#)

[What are common resources of a GLG graph?](#)

[How do I change the number of data samples displayed in a graph?](#)

[How do I change the number of labels and minor ticks?](#)

[How do I set the range of a graph?](#)

[How do I set the initial value for the graphs samples?](#)

[How do I set the initial value for the graph's scrolling time labels?](#)

[How do I supply data to a graph?](#)

[How do I supply custom labels?](#)

[How do I add custom annotations to a graph?](#)

[How do I change the size of the graph's Data Area?](#)

[How do I change the number of lines in a multi-line graph?](#)

[How do I assign line and marker colors in a multi-line graph?](#)

[I have 10 lines, but I can see only 7 line color attributes in a multi-line graph. How do I add more color attributes to the list?](#)

General Widget Questions

What is a GLG Widget?

The **conventional definition of a widget** is a graphical interface element which is usually defined as a reusable piece of code. With this approach, extending a widget's functionality requires adding new code to the widget's source code. To port each widget to a new platform, its source code has to be ported.

GLG Widgets use a different model where **each widget is a GLG drawing** (".g" file) containing a collection of GLG graphical objects that implement the widget's graphical appearance and run-time behavior.

To change the widget, the drawing may be edited with the GLG Graphics Builder, and new custom widget drawings may be created and saved.

At run-time, a widget drawing is loaded and handled by the GLG object engine. The type of the object engine depends on the deployment environment: a C/C++ library for either Windows or Linux/Unix, a Java or C# / .NET class library, a JavaScript / React / Angular / Next.js library for web browser deployment, or an ActiveX control for legacy Windows application.

Since the GLG Widgets have no source code associated with them, their functionality is completely defined by each widget's drawing. Widgets' drawings are cross-platform and may be used by GLG applications on various platforms and programming environments with no porting required. For example, the same widget can be deployed in a C/C++ application on either Windows or Linux/Unix, a Java or C# application, as well as on a web page in a browser.

Where are widget drawings located in the GLG directory?

The widgets' drawings are located in the *widgets* directory of the GLG installation.

For the Community Edition of the Toolkit, the widgets are located in the *DEMOS/widgets* directory. The Community Edition of the GLG Widgets should not be used with the commercial version of the Toolkit.

What do I do with the widget's .g file?

A widget's drawing is [deployed in an application](#) using one of the GLG containers provided by the GLG library for the selected programming environment: C/C++ or X/Qt/Gtk Widget, Java bean, C# control, ActiveX control, Java servlet, ASP.NET HTTP handler, HTML5 JavaScript control or React, Angular and Next.js components, as described in the [next section](#).

A widget's drawing (".g" file) may be [loaded into the GLG Graphics Builder](#) to customize its appearance as described in the [How do I modify a widget?](#) section, or browse the widget's [resources](#), as described in the [How do I browse widget's resources?](#) section. Objects inside the widget drawing can also be [edited directly](#) as described in the [Can I edit objects inside the widget's drawing directly?](#) section below.

How do I use a widget in my application?

At run time, an application uses an appropriate widget container provided by the Toolkit to load and display the widget:

- **X / Motif Widget** provided in the GLG C/C++ library on Unix/Linux platforms
- **Qt or GTK Widget** provided in the GLG C/C++ library on both Windows and Linux/Unix
- **Java Bean** provided in the GLG Class Library for Java (JAR file)
- **C# control** provided in the GLG .NET DLL
- **Java Servlet** provided in the GLG Graphics Server for JSP
- **ASP.NET HTTP Handler** provided in the GLG Graphics Server for ASP.NET
- **ActiveX Control** provided by the GLG OCX component on Windows
- **Windows Control** or an **MFC class** provided by the C/C++ library on Windows
- **Generic cross-platform C or C++ methods** for loading and displaying a drawing provided in the GLG C/C++ library on both Windows and Linux/Unix

Each of the widget containers contains a property that specifies the widget drawing file to be loaded into it. For example, the GLG Java Bean has a ***DrawingFile*** and ***DrawingURL*** properties for supplying a drawing as either a file or URL. Both the X / Motif Widget and ActiveX Control contain a DrawingFile property as well.

After loading the widget drawing, the application uses the *SetResource* API method to set widget's resources and supply dynamic data. There are several versions of the SetResource method:

- *SetDResource* to supply numerical values

- *SetSResource* to supply strings
- *SetGResource* to supply triplets of the XYZ or color RGB values.

In the Java and C# environment, a single overloaded SetResource method with different parameters is used for all value types.

The application may also use the ***Input callback*** to handle widget input. In the Java and C# environments, the callback is implemented as an Input listener.

Most of the widgets are contained in a viewport (window object) named ***\$Widget***.

These widget drawings may be loaded into an application as is. Other widgets, such as process control or electrical symbols, are not contained in a viewport and can't be loaded into an application directly. Instead, these widgets must be added to a viewport-based drawing, and the drawing could then loaded into an application.

Refer to the [programming examples](#) listed in the [next section](#) for samples of the source code.

Refer to the [GLG Programming Reference Manual](#) for more information on using GLG Widgets in C/C++, Java, C#/.NET, as well as JavaScript / React / Angular / Next.js web applications.

Are there programming examples of using GLG Widgets?

Programming examples of using GLG widgets are provided in the following directories:

- examples_c_cpp - C and C++ examples
- examples_java - Java examples
- examples_html5 - HTML5 JavaScript examples
- examples_csharp.NET - C# examples using GLG C# DLL (Windows)
- examples_csharp_ocx - C# examples using GLG ActiveX Control (Windows)
- examples_VB.NET - Visual Basic examples using GLG C# DLL (Windows)
- examples_vbnet_ocs - Visual Basic examples using GLG ActiveX Control (Windows)
- examples_jsp - JSP servlet examples
- examples_ASP.NET - ASP.NET HTTP handler examples (Windows)
- examples_mfc - MFC examples (Windows)
- examples_CPP_CLI - C++ CLI examples using GLG C# DLL (Windows)

GLG Demos provide more elaborate examples of using the Toolkit and contain demo source code in the following directories:

- DEMOS - C / C++ demos
- DEMOS_JAVA - Java demos
- DEMOS_C# - C# / .NET demos (Windows)
- DEMOS_HTML5 - HTML5 & JavaScript web demos

Refer to the [GLG Programming Reference Manual](#) for more information on using GLG Widgets in C/C++, Java and .NET applications.

How do I load a widget in the Graphics Builder?

The most convenient way of loading a new widget into the Graphics Builder is using its Widget Palettes. To pop up a widget palette in the Builder, click on *Palettes* in the main menu, then select a desired palette type - Dials and meters, Real-Time Charts, etc.

To replace the current drawing with the widget's drawing, press and hold the *Control* key, click on the widget icon in the palette and click on *OK* to load the widget drawing. This is the **preferred way** of loading a single widget into the Builder for editing, since it's also loads an animation command for prototyping the widget with data.

After editing, save the modified widget into a file in your project directory. The modified drawing can be loaded in the Graphics Builder using the *File, Open* menu option.

To add a widget from a palette into the current drawing (which may already contain several other widgets), simply click on the widget icon with the mouse. To add several widgets, click several times and position each widget in the drawing. The resulting dashboard drawing with several widgets can be loaded into an application as a single entity.

When a widget is added to the drawing, it is assigned a default name: *\$Widget* for widgets contained in a viewport, or *\$Drawing* for widgets without a rectangular viewport container. When adding several widgets to the drawing, you may want to assign a distinct name to each widget. This will make it possible to access widgets in the drawing via [resources](#), using the widget name in the resource path.

How do I modify a widget?

The widget can be modified by [loading](#) its drawing in the Graphics Builder, which allows you to edit the drawing and save the customized version of it.

The most commonly used widget properties can be accessed via the *Public Properties*



dialog. The *Public Properties* dialog displays public properties defined inside the

widget and is different from the *Properties*  dialog, which displays only the attribute properties of the widget's viewport.

To edit widget properties, click on the widget in the drawing to select it, then either use the *Object, Public Properties* menu option or click on the *Public Properties*  toolbar button to display the dialog.

The properties in the dialog are grouped into categories, with buttons to access properties in each category. Each property can be edited directly by entering a value in a text box next to the property. An ellipsis  button next to each property provides access to a dialog that has more choices for editing the property, including palettes or menus.

The widget's drawing can also be edited via [resources](#) or by [direct editing](#) of the widget's drawing, as described in the following sections.

While public properties provide access to the most commonly used widget properties, resources provide access to all widget properties. Resources can also be used at **run time** to change widget properties via an API.

Refer to the [GLG Widgets Reference Manual](#) for detailed information on editing GLG widgets.

How do I browse widget's resources?

The Toolkit's resource mechanism provides a universal way for accessing properties of any object inside a GLG drawing or a GLG widget. Resources are hierarchical, making it possible to browse not only the objects in the drawing, but also subobjects and properties inside each object.

To browse resources of a widget, [load](#) the widget into the Graphics Builder, then click on the widget with the mouse to select it. The name of the selected widget will be displayed in the *Name:* field below the drawing area.

Click on the *Resources*  button in the toolbar to bring the Resource Browser. Use tooltips to identify toolbar buttons. The Resource Browser may also be activated via the *Object, Resources* menu option.

The Resource Browser will show a list of the selected widget resources. Selecting a resource in the list, such as *FillColor*, with the mouse activates a dialog containing menus and palettes for editing the resource's value.

Some resources, such as a dial widget's *ActiveElement*, have the ">>" symbols appended at the end, indicating that they contain more resources inside. Double-click on such resources to display resources inside them. To go back, double-click on the ".." entry at the beginning of the resource list. As an alternative to double-clicking on a list item, the *Select* button at the bottom of the Resource Browser may be used while the list item is selected.

The Resources Browser displays resources of the selected object. Selecting a different object show its resources. If no object is selected, the Resource Browser lists resources

of the whole drawing. If a drawing contains one or more widgets, the Resource Browser will show the widgets as resources, listing them by name. Double-clicking on a widget name in the list will show resources inside that widget.

When creating a drawing containing several widgets, it is important to name the widgets using distinct names, so that the widget parameters could be accessed as resources using the API at run time.

Can I edit objects inside the widget's drawing directly?

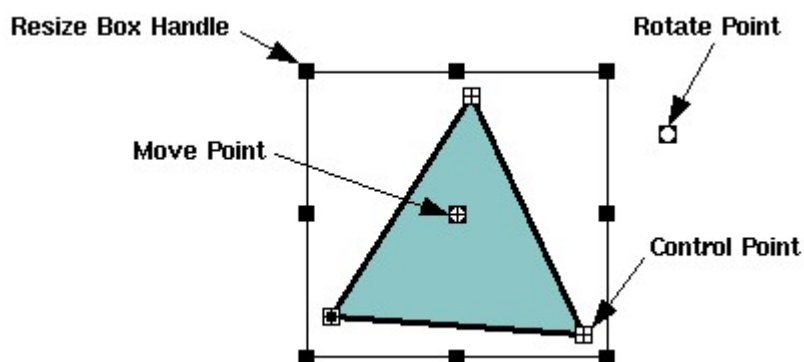
A widget drawing may be edited the same way as any other GLG drawing.

Widgets such as dials, meters and charts are contained in a viewport object named *\$Widget* which provides the widget with its own window. To edit objects inside the widget viewport, select the widget with the mouse and click on the *Hierarchy Down*



button on the lower left side of the drawing area, or use the *Traverse, Hierarchy Down* menu option. This will bring you "down" into the widget viewport, so that you can edit objects inside it.

When you get inside of the viewport, you can select and edit objects in it. To edit an object's attributes, click on the object to select it, then adjust its geometry by clicking and dragging its control and resize points. To edit point coordinates, *Shift-click* on a control point by pressing and holding the *Shift* key, then clicking on the point with the mouse. The following picture shows object control points:



The *Properties*  toolbar button activates the Object Properties dialog for editing object attributes. Use the ellipsis  button next to a property to display palettes and dialogs for editing the property.



The *Edit Toolbox* toolbar button activates the Edit Toolbox for editing object

colors and other attributes. The *Layout Toolbox* toolbar button activates the Layout Toolbox with options for aligning groups of objects. A group may be created by dragging a rectangle with the mouse or *Ctrl-clicking* on objects in the drawing. Every *Ctrl-click* adds or deletes an object from the temporary editing group.



Use the *Hierarchy Up* button to go back to the top level of the drawing.

Some widgets, such as process control or electrical symbols, do not have a viewport and use a container object named *\$Drawing*. If the widget does not have a viewport, the *Hierarchy Down* button can still be used in the same way to get access to objects inside the widget's container object.

Can I add new objects to a widget or delete the ones I do not need?

Yes, you can. [Select the widget and go down into its viewport](#) as described in the previous section to get access to objects inside of the widget's viewport. Use the Object Palette on the upper left side of the drawing area to draw new objects.



To delete objects, select the object to be deleted and press the *Cut* toolbar button.

How do I animate a widget in the Builder?

The Run mode of the Graphics Builder is used to prototype a widget's run time behavior. The Run mode is started by clicking on the *Start* toolbar button, entering an animation command and pressing OK. The *Run, Start* menu option can be used instead of the *Start* toolbar button. To stop the animation, click on the *Stop* toolbar button or use the *Run, Stop* menu option.



All widget drawings include a matching animation command to animate them with changing data. To load a widget together with its animation command, use *Ctrl-click* when selecting a widget from a palette. This will discard the current drawing and load the widget's drawing. The widget can then be prototyped by simply pressing the *Start*



toolbar button and pressing OK to accept the animation command. Click on *Stop*



to stop animation.

Dials, sliders and other input widgets may be animated without a run command by **dragging their active elements with the mouse**. Click on *Start* toolbar button, press *Skip Command*



to start prototyping without a command, then click and drag the dial or slider with the mouse. Click on *Stop*  to stop.

To disable user interaction for dials that will be used for output only, empty the *Handler* text box in the dial viewport's properties. This will delete the widget's interaction handler.

Refer to the *Data Generation Utility* section in the *GLG Programming Tools and Utilities* chapter of the [GLG Programming Reference Manual](#) for more information on the run command options.

How do I supply data to animate a widget at run time?

There are two ways to supply data for run-time animation, using either [resources](#) or [tags](#).

To use resources, the application can use [SetResource](#) API methods to supply data using resource paths, such as "Dial1/Value" or "Dial2/Value", for example:

```
SetDResource( drawing, "Dial1/Value", 50. );
```

When using resources, the application would need to know the exact resource paths used to supply value. The [Resource Browser](#) inside the Graphics Builder can be used to browse resources. The resource paths observed in the Resource Browser can then be used in the code to supply data.

GLG tags provide an alternative way for supplying data that frees an application from a need to know a drawing's resource hierarchy and the exact resource paths. The tag-based animation makes it possible to write generic code that can animate an arbitrary drawing without knowing its content.

A data tag can be attached to a widget property and used for animating it with data. A tag has a *TagSource* property which defines the name of the data source variable that will supply dynamic data for updating the widget at run time.

For example, load a dial widget in the Graphics Builder using the [steps described above](#), then select the widget by clicking on it with the mouse and open its [Public Properties dialog](#).

There will be a small button with the letter "T" next to the widget's *Value* property, indicating that the property has a data tag attached. Clicking on the button opens a *Data Tag* dialog. Entering "Value1" in the *TagSource* field and saving the drawing will allow the application to animate this widget property by using the *SetDTag* API method, for example:

```
SetDTag( drawing, "Value1", 50. );
```

In a real application, *TagSource* is set to an identifier what defines a source of data, such as a process data base tag.

The source code of the GLG *SimpleViewer* provides an example of an application that loads arbitrary drawings and updates them with data using tags without knowing the content of the drawing. When a drawing is loaded, the application queries the list of data tags defined in the drawing and uses it for data animation. For each tag, the application obtains data from a corresponding process database tag defined by the tag's *TagSource*, and updates the tag in the drawing using the *SetDTag* API call.

Refer to the *Animating Widgets with Data Using Tags* chapter of the [GLG Builder and Animation Tutorial](#) for a detailed example of using data tags.

How do I create a new widget from scratch?

Since widgets are GLG drawings, creating a new widgets is as simple as creating and saving a new drawing using the GLG Graphics Builder.

A GLG widget must be contained in a viewport named *\$Widget*. The viewport object encapsulated a native window and provides the widget with a drawing surface. At run time, the *\$Widget* viewport is loaded and displayed in an application using one of the provided [native containers](#)

To create a new widget, select one of the *File, New, Widget* menu options. This creates a new *\$Widget* viewport that will contain the widget's drawing, and brings you down into it, so you can start creating objects. **Constraints, Dynamics** and **Input Handlers** may be used to define the widget's run time behavior and wire complex relationships between objects in the widget's drawing.

To create a new dial widget, it may be easier to start with an existing dial widget as a template and modify it. For example, to [create a dial with two needles](#), use a dial widget with one needle and add the second needle by copying the needle or creating a new needle from scratch and attaching rotation dynamics to it. In either case, the dial's angular axis with ticks and labels, as well as the interaction handler setup are reused.

Refer to the [GLG Builder and Animation Tutorial](#) for more information on creating dynamic drawings.

Refer to the *Input Objects* chapter of the [GLG User's Guide and Builder Reference Manual](#) for more information on input handlers.

I need to display several widgets. Do I need a separate Bean / Control container for each widget?

Instead of using a several containers, one for each widget, you can [create a composite drawing](#) containing several widgets (a dashboard) and use a single container to display it in an application. Not only it results in more efficient use of resources, but also the layout of widgets in the dashboard may be [interactively edited](#) using the Graphics Builder.

How do I create a dashboard containing several widgets?

To create a dashboard, first create a new drawing using one of the *File, New, Widget* menu options. This will create a [viewport named \\$Widget](#) and will go down into it. Click on *Palettes* in the main menu and select a desired palette type, such as dials and meters.

Click on a widget icon in the palette with the mouse to insert the widget into the drawing. Position the widget and bring its [Public Properties dialog](#), then change widget's *Name* from *\$Widget* to a unique custom name, such as *Widget1*. Edit widget properties displayed in the dialog to customize the widget. As a minimum, set the dial's *Low* and *High*, and set *Units* of the *Unit Label*.

Repeat the previous steps to add several widgets to the drawing, positioning and naming each one *Widget2*, *Widget3*, and so on.

When finished, use the *Hierarchy Up*  button to go to the top level to see the finished dashboard and save the drawing.

Use the [Resource Browser](#) to browse widgets' resources. When the top *\$Widget* viewport is selected, the Resource Browser will show the *Widget1*, *Widget2*, ... resources. Double-clicking on each entry will show resources inside it. When browsing resources, the resource paths of selected resources are displayed at the top of the Resource Browser. These paths are used in an application to access resources of each widget at run-time using the GLG's *GetResource* and [SetResource](#) methods.

Dials and Meters FAQ

What is a GLG dial widget?

Like [any other GLG widget](#), a GLG dial is a GLG drawing representing a dial with one or more rotating needles, an angular axis with ticks and labels, and an optional label displaying the dial's current value. The objects in the dial's drawing are wired up using the GLG dynamics and constraints to implement a dial widget.

How do I edit a dial widget's properties?

Any dial widget may be [loaded](#) in the GLG Graphics Builder and customized by changing its properties. A customized version of the widget can then be saved in a new drawing file that will be used by an application.

The [Public Properties dialog](#) is the primary way to modify a widget in the Graphics Builder, as described in the [How do I modify a widget?](#) section.

Widget properties can also be accessed via [resources](#) as described in the [How do I browse widget's resources?](#) section. Resources can be used to access widget properties via API at run time.

Elements of a dial can be also be [edited directly](#) to change the dial's needle geometry, position the dial's unit and value labels, or add custom annotations.

Refer to the *Meters, Dials and Avionics Gauges* chapter of the [GLG Widget Reference Manual](#) for more information on dial widget properties.

How do I prototype a dial widget in the Graphics Builder?

The Run mode of the GLG Graphics Builder may be used to prototype the dial's run-time behavior using simulated data as described in the [How do I animate a widget in the Builder?](#) section.

How do I use a dial widget in my application?

A dial widget is deployed in an application using one of the programming containers provided in the GLG libraries, as described in the [How do I use a widget in my application?](#) section. An application uses an API to supply data displayed in the dial using either [resources](#) or [tags](#) as described in the [How do I supply data to animate a widget at run time?](#) section.

How do I handle the dial widget's input?

Dial widgets have an attached interaction handler that handles user interaction. A user can change a dial's value with the mouse, by either clicking on the new desired needle position, or by dragging the needle with the mouse. This behavior can be prototyped in the Graphics Builder as described in the [How do I animate a widget in the Builder?](#) section.

When a dial widget is deployed in an application at run-time, the application's *Input* callback is invoked every time the dial is moved with the mouse. A message object provided as one of the parameters of the *Input* callback provides information about the dial (there may be several dials in the drawing), its value and the user action.

Refer to the [GLG Programming Reference Manual](#) for more information on handling user interaction in various programming environments (C/C++, Java, C#/.NET, JavaScript / React / Angular / Next.js, etc.).

Refer to [demos and examples](#) for samples of source code that handle user interaction.

I want to use a dial widget for output only. How do I disable the dial's input?

To disable user interaction for dials that will be used for output only, empty the *Handler* text box in the dial viewport's *Properties*  dialog. This will delete the widget's interaction handler.

How do I add more than one needle to a dial?

While the GLG Dials and Meters palette provides a few dials with two needles, additional needles can also be added to other dial widgets.

The easiest way to add an additional needle to a dial is by copying and pasting a copy of the existing needle:

1. Select the dial's *\$Widget* viewport and click on the *Hierarchy Down*  button to [traverse the hierarchy down](#), inside the widget's viewport.
2. Click on the needle to select it. The *ActiveElement* name will be displayed in the selected *Name* field at the bottom of the drawing area.
3. Click on the *Copy*  toolbar button, then paste using the *Paste*  button. The new copy of the needle will be selected.
4. Click on the *Properties*  toolbar button to edit properties of the new needle and change its name to *ActiveElement2*.
5. Save the drawing.

Select the dial's *\$Widget* viewport and click on the *Resources*  toolbar button to bring the Resource Browser: it will show an *ActiveElement2* entry which is the second needle. Double-click on it to display the *Value*, *Low* and *High* resources inside it. These resources will control the second needle. For example, *ActiveElement2/Value* may be used to supply the value for the second needle.

The *Low* and *High* resources of the second needle may be constrained to the *Low* and *High* of the first needle if so desired. Follow the steps below to constrain the *Low* resource:

1. Select the *ActiveElement2/Low* resource in the Resource Browser.
2. Click on the *Constrain All* button in the *Resource Object* dialog, then click on the *Use Resource* button.
3. Navigate to the *Low* resource of the first needle (*\$Widget/ActiveElement/Low*) and press *Select* at the bottom of the Resource Browser.

Repeat steps 1-3 for the *ActiveElement2/High* resource, to constraint it to *ActiveElement/High*.

You can paste any number of additional needles following the steps above.

To eliminate the need to constrain the *Low* and *High* resources for each needle, set their *Global* flag to GLOBAL before copying the needle and select the *Options, Paste Clone Type, Strong Clone* in the main menu. This will automatically constrain the *Low* and *High* resources of the pasted copy to the corresponding resources of the original needle when the *Copy* and *Paste* operations are performed.

Chart and Graph Widgets, General Questions

How do I use a chart or a graph widget in my application?

A chart or a graph widget is deployed in an application using one of the programming containers provided in the GLG libraries, as described in the [How do I use a widget in my application?](#) section. An application uses an API to supply data displayed in the chart using either [resources](#) or [tags](#) as described in the [How do I supply data to animate a widget at run time?](#) section.

How do I prototype a chart or a graph in the Graphics Builder?

The Run mode of the GLG Graphics Builder may be used to prototype the chart's run-time behavior using simulated data as described in the [How do I animate a widget in the Builder?](#) section.

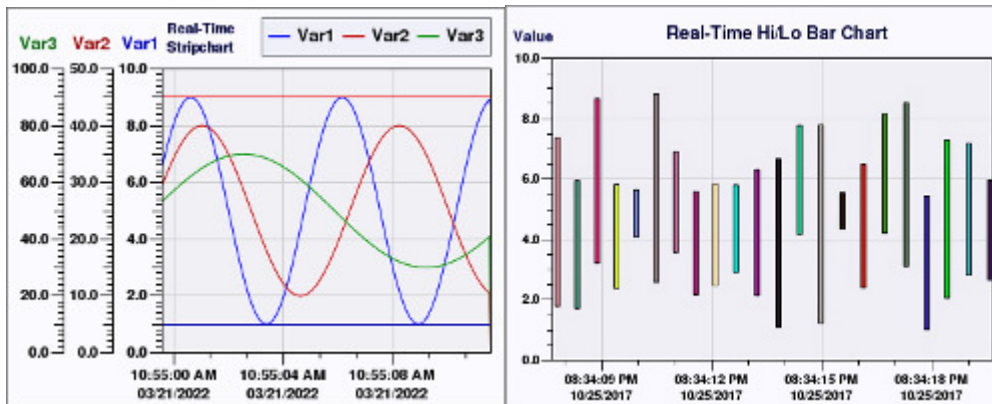
Real Time Charts

What is the difference between the Real-Time Charts and the Graph Widgets

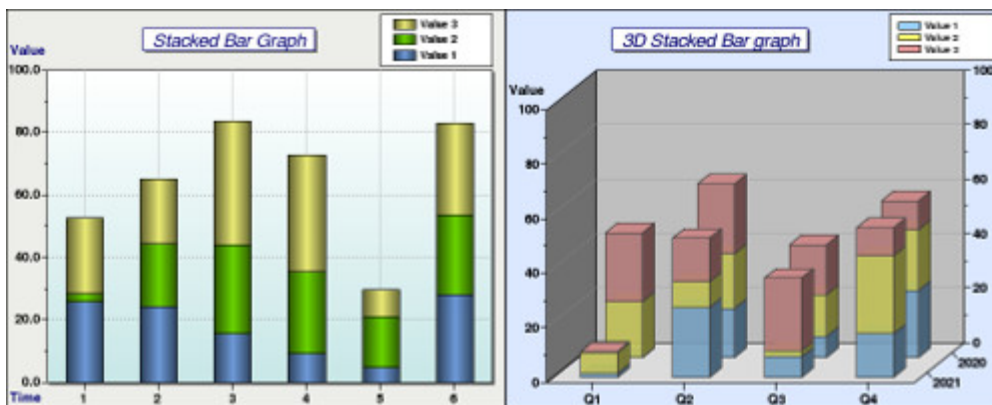
The Real-Time Charts are optimized to render charts with large number of data points and fast update rates, and use an integrated *Chart* object for fast rendering with support for integrated zooming and scrolling, chart tooltips, cursor feedback and point selection.

The 2D Graphs and 3D Graphs widget sets contain legacy 2D and 3D graph widgets that provide a variety of graph types, such as bar, pie, polar and other graphs with presentation-style graphics.

While the Real-Time Charts are implemented using a single performance-optimized *Chart* object with an integrated data buffer, the 2D and 3D graphs (including their axes and legends) are constructed using a collection of individual graphical objects, such as polygons, text, series and other objects. The 2D and 3D graphs support gradient shading and 3D bar rendering, as well as packed, stacked and other multi-set bar graphs.



Real-Time Charts



2D and 3D Graphs

How do I edit properties of the Real-Time Chart widget

A real-time chart widget may be [loaded](#) in the GLG Graphics Builder and customized by changing its properties. A customized version of the widget can then be saved in a new drawing file that will be used by an application.

The [Public Properties dialog](#) is the primary way to modify a widget in the Graphics Builder, as described in the [How do I modify a widget?](#) section.

The behavior and appearance of a real-time chart is controlled by the properties of the Chart object contained inside the real-time chart widget's viewport. The *Edit Chart* button in the [Public Properties dialog](#) dialog provides access to the attributes of the Chart object. Refer to the *Chart Objects* chapter of the *GLG User's Guide and Builder Reference Manual* for a detailed description of the chart's attributes.

The [Public Properties dialog](#) also contains buttons for editing attributes of the chart's background, title and legend (if any), as well as the offsets that control the chart layout margins.

Chart properties can also be accessed via [resources](#) as described in the [How do I browse widget's resources?](#) section. Resources can be used to access chart properties via API at run time.

Elements of a chart widget can also be [edited directly](#) by going "down" into it using

the [Hierarchy Down](#)  button. While inside the widget viewport, elements of the chart can be edited with the mouse to change their geometry, edit their attributes or add custom annotations.

Refer to the *Real-Time Chart Widgets* chapter of the [GLG Widget Reference Manual](#) for more information on using the real-time charts widgets.

How do I supply data to the Real-Time Chart

An application supplies data to the real-time chart widgets using either [resources](#) or [tags](#), as described in the [How do I supply data to animate a widget at run time?](#) section.

The *Chart* object inside the real-time chart widget is responsible for rendering the chart. An application supplies data for a chart using the Chart's *ValueEntryPoint*, *TimeEntryPoint* and *ValidEntryPoint* described in the *Chart Objects* chapter of the [GLG User's Guide and Builder Reference Manual](#). The data can be supplied to the entry points using either [resources](#), or using [tags](#) attached to the entry points.

An application can either provide a time stamp for every data sample, or let the chart use current time as a time stamp. The chart automatically positions and scrolls its data samples according to their time stamps. The chart also supports the use of invalid data samples, which are displayed as gaps in the plot lines.

The chart maintains an internal data buffer that makes it possible to scroll and zoom data accumulated in the chart using integrated zooming and scrolling.

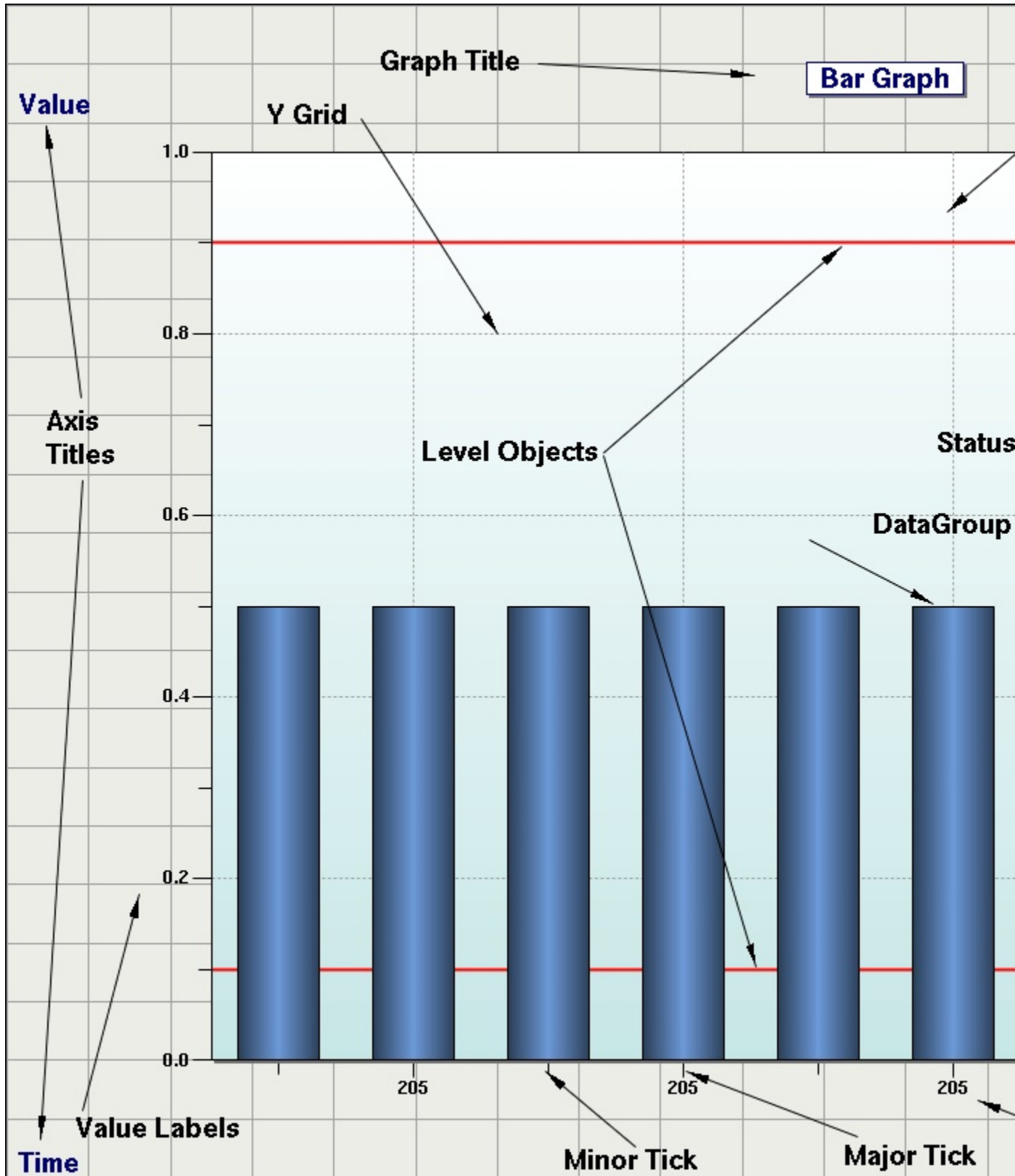
2D and 3D Graphs

What is a GLG graph widget?

Like [any other GLG widget](#), a GLG graph is a GLG drawing representing a bar, line or some other graph widget. The objects in the drawing are wired up using the GLG dynamics, history and constraints to behave like a scrolling graph when filled with data. The resources defined in the drawing are used by an application at [run-time](#) to configure the graph and supply graph data.

Drawings of all GLG graphs are constructed using similar elements to represent graph components such as axes, ticks, labels and data. A typical graph has a data area, X and Y axes, grids, titles, minor and major ticks, major tick labels for each axis, a data group

containing graph data samples and other elements as shown in the picture below. [Click on the picture](#) to see a bigger image.



How do I modify graph's resources and geometry?

Any graph widget may be [loaded](#) in the GLG Graphics Builder and edited as described in the [How do I modify a widget?](#) section.

Unlike other GLG widgets that can be edited via the *Public Properties* dialog, the legacy 2D and 3D graph widgets are edited only via [resources](#). The Resource Browser may be used to [browse and modify a graph widget's resources](#).

Elements of a graph may be also be [edited directly](#) as described in the [Can I edit objects inside the widget's drawing directly?](#) section to position graph's titles, change geometry of the graph's *DataArea* or add custom annotations.

What are common resources of a GLG graph?

All GLG graphs have similar resource hierarchies, which slightly differ depending on the graph type.

The following lists resources of a bar graph. Most of these resources are present in all graphs regardless of their type:

- *DataGroup/Factor* - defines the number of datasamples in the graph
- *DataGroup/DataSample/Low* - defines the low range of data
- *DataGroup/DataSample/High* - defines the high range of data
- *DataGroup/DataSample/Value* - defines an initial value for graph's samples
- *DataGroup/ScrollType* - defines the graph's scroll type (WRAPPED or SCROLLED)
- *DataGroup/EntryPoint* - an entry point for pushing data values into the graph.
- *XLabelGroup/Factor* - defines the number of X labels, major ticks and grids
- *XLabelGroup/MinorFactor* - defines the number of minor ticks per each major tick interval
- *XLabelGroup/XLabel/String* - defines an initial value for the label, usually set to an empty string
- *XLabelGroup/EntryPoint* - an entry point for pushing X axis labels into the graph
- *YLabelGroup/Factor* - defines the number of Y labels, major ticks and grids
- *YLabelGroup/MinorFactor* - defines the number of minor ticks per each major tick interval
- *YLabelGroup/Low* - defines the low range of the Y axis labels, same as *DataGroup/DataSample/Low*
- *YLabelGroup/High* - defines the high range of the Y axis labels, same as *DataGroup/DataSample/High*

A line graph has a slightly different resource hierarchy for its *DataGroup*:

- *DataGroup/Factor* - defines the number of datasamples in the graph
- *DataGroup/Marker/DataSample/Low* - defines the low range of data

- *DataGroup/Marker/DataSample/High* - defines the high range of data
- *DataGroup/Marker/DataSample/Value* - defines an initial value for graph's samples
- *DataGroup/ScrollType* - defines the graph's scroll type
- *DataGroup/EntryPoint* - an entry point for pushing data values into the graph

A multi-line graph has a different *DataGroup* hierarchy as well:

- *DataGroupOne/Factor* - defines the number of lines in the graph
- *DataGroupOne/DataGroup/Factor* - defines a number of point in each line
- *DataGroupOne/DataGroup/Marker/DataSample/Low* - defines the low range
- *DataGroupOne/DataGroup/Marker/DataSample/High* - defines the high range
- *DataGroupOne/DataGroup/Marker/DataSample/Value* - defines an initial value
- *DataGroupOne/DataGroup/ScrollType* - defines the graph's scroll type
- *DataGroupOne/EntryPoint* - an entry point for pushing data values into all lines
- *DataGroupOne/DataGroupN/EntryPoint* - an entry point for pushing values into the N-th line

The top level *DataGroupOne/EntryPoint* resource of a multi-line graph may be used to push values into all lines of the graph. The entry points of each line (the *DataGroupOne/DataGroupN/EntryPoint* resources, where *N* is the index of a line) may be used to push datasamples into each line separately.

You can use the Resource Browser to interactively [browse a graph's resources](#).

Refer to the *2D and 3D Graphs* chapter of the [GLG Widgets Reference Manual](#) for more information on graph's resources and their usage.

How do I change the number of data samples displayed in a graph?

For single-series graphs, such as a bar or line graphs, the *DataGroup/Factor* resource defines the number of the graph's datasamples.

For multi-series graphs, such as a multi-line graph, the *DataGroupOne/DataGroup/Factor* resource defines the initial number of datasamples in each line. The number of datasamples in each of the graph's line may also be changed individually, by setting the *DataGroupOne/DataGroupN/Factor* resource of each line after the hierarchy setup.

You can use the [Resource Browser](#) inside the Graphics Builder to interactively [browse resources of graph widgets](#).

How do I change the number of labels and minor ticks?

The number of labels and ticks on the X axis may be changed using the following resources:

- *XLabelGroup/Factor* - defines the number of X labels, major ticks and grids

- *XLabelGroup/MinorFactor* - defines the number of minor ticks per each major tick interval

For the Y grid:

- *YLabelGroup/Factor* - defines the number of Y labels, major ticks and grids
- *YLabelGroup/MinorFactor* - defines the number of minor ticks per each major tick interval

You can use the [Resource Browser](#) inside the Graphics Builder to interactively [browse resources of graph widgets](#).

How do I set the range of a graph?

For single-series bar graphs, the following resources define the graph's range:

- *DataGroup/DataSample/Low* - defines the low range of data
- *DataGroup/DataSample/High* - defines the high range of data

For single-series line graphs, the resource are slightly different:

- *DataGroup/Marker/DataSample/Low* - defines the low range of data
- *DataGroup/Marker/DataSample/High* - defines the high range of data

For multi-line graphs, the resources are:

- *DataGroupOne/DataGroup/Marker/DataSample/Low* - defines the low range
- *DataGroupOne/DataGroup/Marker/DataSample/High* - defines the high range

You can use the [Resource Browser](#) inside the Graphics Builder to interactively [browse resources of graph widgets](#).

How do I set the initial value for the graphs samples?

The *Value* resource of the *DataSample* template defines the initial value of the graph's datasamples before the data are filled in. The following resources may be used to set it depending on the graph type:

- *DataGroup/DataSample/Value* - for single-series bar graphs
- *DataGroup/Marker/DataSample/Value* - for single-series line graphs
- *DataGroupOne/DataGroup/Marker/DataSample/Value* - for multi-line graphs

The resource must be set before the hierarchy setup at run time, or set in the graph's drawing.

You can use the [Resource Browser](#) inside the Graphics Builder to interactively [browse resources of graph widgets](#).

How do I set the initial value for the graph's scrolling time labels?

The initial value of the scrolling labels is set using the label's *String* resource. For graphs that scroll horizontally along the X axis, the *XLabelGroup/XLabel/String* resource may be used.

The resource must be set before the hierarchy setup at run time, or set in the graph's drawing.

How do I supply data to a graph?

A graph provides entry points for supplying scrolling graph data and labels. For example, in a single-series graph such as a bar or line graph, the following resources may be used to **supply data** and **scrolling time labels** to the graph:

- *DataGroup/EntryPoint* - graph data entry point
- *XLabelGroup/EntryPoint* - label entry point

For multi-line graphs, the *DataGroupOne/EntryPoint* may be used to supply data to all lines of the graph at once. Individual entry points, such as *DataGroupOne/DataGroupN/EntryPoint*, where *N* is a line index, may be used to supply data to each line separately.

How do I supply custom labels?

Custom labels for the scrolling time axis are supplied through the *XLabelGroup/XLabel/EntryPoint* resource for graphs with the horizontal time axis, as described in the [previous section](#).

The labels for the value axis are generated automatically based on the range of the graph. The number of digits after the decimal point may be specified using the label's *Format* resource, which defines a C-style format. To overwrite automatic labeling and supply custom values, the *Format* resource may be set to a string which does not contain the "%" format symbol.

For example, the format string "%.0lf" will display automatic labels with no digits after the decimal point. Setting the *YLabelGroup/YLabel0/Format* resource to the "3.75" string will cause the string to be displayed in the first Y label. The *Global* flag of the *YLabelGroup/YLabel/Format* resource must be set to LOCAL in the drawing to set the *Format* resource of each label to a different value.

Refer to the [GLG Widgets Reference Manual](#) for more information.

How do I add custom annotations to a graph?

Since the graph widget is a GLG drawing, you can add new objects to it as described in the [Can I add new objects inside a widget or delete the ones I do not need?](#) section.

Text objects may be added to annotate the graph or its axis. A label may be added to each of the graph's datasamples by adding a text object to the datasample's template to display the sample's value.

How do I change the size of the graph's Data Area?

The size and position of the *DataArea* or any other element of the graph may be changed by editing it using the mouse as described in the [Can I edit objects in the widget's drawing directly?](#) section.

After [traversing the hierarchy down](#) into the graph widget, click on the *DataArea* to select it and adjust its size and position by dragging the *DataArea* or its control points with the mouse.

When selecting the *DataArea*, notice the its name appearing in the selected *Name:* indicator at the bottom of the Graphics Builder's drawing area. Since the graph contains many objects on top of the *DataArea*, the mouse click may select one of this object instead of the *DataArea*.

If you have problems selecting *DataArea*, press and hold the *Shift* key while clicking on the *DataArea*. This will bring the *Object Selection* dialog if several objects are potentially selected by the click, so that you can scroll through all selected objects to select the *DataArea*. If the *Properties* dialog is active, the arrows to scroll through all selections will become active in the upper left corner of the dialog, instead of activating the *Object Selection* dialog.

How do I change the number of lines in a multi-line graph?

The number of lines is controlled by the graph's *DataGroupOne/Factor* resource.

How do I assign line and marker colors in a multi-line graph?

The *DataGroupOne/DataGroup/Polygon* resource is a line template, and it has *Color0*, *Color1*, ... resources that define the color of each line. To change a color, select the color resource in the [Resource Browser](#), then select a new color from a color palette. The *DataGroupOne/DataGroup/Marker* resource defines a template for line markers. It also has *Color0*, *Color1*, ... resources that define marker color for each line, which may be changed using the [Resource Browser](#).

I have 10 lines, but I can see only 7 line color attributes in a multi-line graph. How do I add more color attributes to the list?

The line colors are defined in a color list transformation attached to the *EdgeColor* of the line polygon object. By default, the color list contains seven colors. The number of colors in the list may be increased by editing the color list transformation.

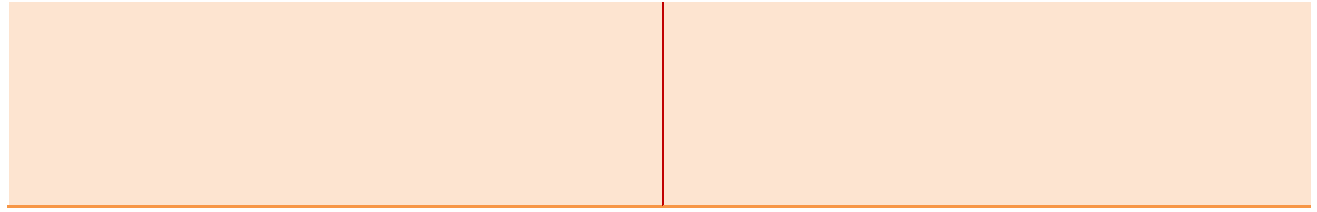
To increase the number of line colors:

1. Select the graph's *\$Widget* viewport and click on the *Hierarchy Down*  button to [traverse the hierarchy down](#) into the graph.

2. Select the line series named *DataGroupOne* (a line with markers) at the bottom of the *DataArea*, and [traverse the hierarchy down](#) into it.
3. You'll see the *DataGroup* object, which is a polyline object used as a line template. Click on the polyline to select it.
4. Click on the *Properties*  toolbar button to display its *Properties* dialog.
5. Click on the ellipsis  button next to the *Polygon* attribute.
6. Notice a small "X" button next to the *EdgeColor* attribute that indicates it has a transformation attached to it. Click on the button to edit the color list transformation attached to the *EdgeColor*.
7. Click on the *List of Values* button to display the list of colors. Select the last color in the list, then click on the *Add* button several times to add a desired number of colors.
8. Select each of the added color entries, name it *ColorN*, where *N* is a sequential item index: *Color8*, *Color9*, and so on. Set the *Global* flag of each color entry to GLOBAL.
9. Save the modified graph.

The marker colors are defined by a color list transformation attached to the *FillColor* attribute of the marker template. To increase the number of marker colors, follow the following steps:

1. Perform the steps 1-3 listed above to select the *DataGroup* object used as a line template.
2. Click on the *Hierarchy Down*  button to [traverse the hierarchy down](#) into the *DataGroup*.
3. Click on the marker template (marker object) to select it, then click on the *Properties*  toolbar button to display the marker's *Properties* dialog.
4. Notice a small "X" button next to the *FillColor* attribute that indicates it has a transformation attached to it. Click on the button to edit the color list transformation attached to the *FillColor*.
5. Perform the steps 7-9 listed above to add colors to the color list transformation.



Note: ** Specifications are copied from manufacturer data. Please contact before buying.