

Adobe Graphics Server and Illustrator

Produce Dynamic Graphs

Adobe Graphics Server and Illustrator

Produce Dynamic Graphs

Contents

Illustrator Sets the Stage	1
Many Possibilities	2
Graphing: A Simple Example	2
Variables Make Graphs Dynamic	5
Applying Data to the Template	6
XML Graph Display Commands	7
Graphing: A Second Example	8
Automation Advantage	10



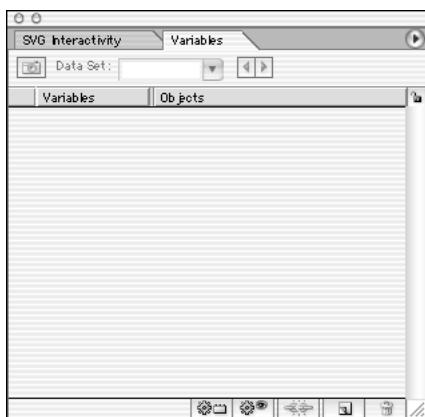
Adobe Graphics server and Illustrator

Produce Dynamic Graphs

*By Mike Cirioli and Jason Wehling
Version 1: 2002-04-24*

Companies are inundated with data—internal data, external data, customer data, sales data. How can a company organize this information and present it visually? Often, companies want to include visual presentations of data as graphs in Web pages, annual reports, and other company documents. In the past, the process was necessarily manual, and therefore time-consuming and expensive. Adobe Graphics Server provides a way to automate the production of graphs from templates created in Adobe Illustrator.

With Graphics Server and Illustrator 10, you can create a wide variety of dynamically generated graphs. Designers can create beautiful chart templates in a wide variety of styles using design tools that they are comfortable with—namely, Illustrator. Then, a programmer can use Graphics Server to apply dynamic data values to those templates, and render the output in multiple formats and resolutions. Graphics Server achieves this capability with the SVG parsing engine and Illustrator's Variables palette.



Illustrator Sets the Stage

Using Illustrator, a designer first chooses a graph type from the Charts palette and drags an instance of it onto a document. This step creates the basic chart object that will be customized. At this point, the designer can create sample data columns and labels, change colors and fonts, and perform other transformations on the chart object. A good idea is to create the both the minimum and maximum number of sample data columns that appear in the chart. Because Graphics Server can automatically change the scale of the graph axis to accommodate a varying number of data points, creating sample columns gives you a good idea how a graph appears as the data changes. If you don't plan to have a varying number of data columns, the axes are scaled the same as they appear in Illustrator, unless you change them before rendering by modifying the `<format>` element of the SVG template. (But we'll cover more about the XML-specifics in a moment.)

After the basic look and feel of the graph is created, the designer must select the graph and then choose to make it dynamic by using Illustrator's Variables palette. This step assigns an SVG variable to the graph's data, and Graphics Server uses this variable to render the graph with the dynamic data. Because you can't directly assign variables to many of the elements that constitute the appearance of a chart, Graphics Server can't change many of these attributes. This limitation can be overcome by several approaches.

An SVG template is basically an XML description of an Illustrator document. As such, you can use a wide variety of tools to manipulate graph descriptions. Graphics Server expects a graph description to be embedded within a <switch> element. The graph foreign object is contained within one branch of the <switch> tag, and the child elements of the <switch> tag describe both the data and appearance of the graph. Within the switch statement are two child elements, <data> and <format>. These two elements are the key to producing dynamic graphs with Graphics Server. The <data> element contains the information regarding the data values for the graph, while the <format> element describes the type and appearance of the graph.

Note: For larger scale changes, you might prefer to modify the template by using an XML parsing library. The Xerces Java library is popular, or you can even use a regular XML or text editor to make changes manually.

Many Possibilities

You need to determine to what degree an application will control the graphs, and you can use a couple of different approaches. You can make all changes to the look and feel within Illustrator, and your application determines which graphs to use as templates for the various data sets. Or, you can create fewer base templates and use XML tools to further manipulate the look and feel.

With the first approach, the graph and its look and feel are created within Illustrator. This approach works well for graphs that don't vary in appearance, except for the number and values of the data points. Using Graphics Server, you can then apply new data to the existing template with the <loadContent...> and <applyVariables...> commands, and render the output in the desired format.

The second approach involves creating a graph in Illustrator, saving it as SVG, and then writing code that can modify or replace the <format> elements of the XML in the SVG file. After you modify the XML for the template, you use Graphics Server in the same manner as the first approach—you apply new data values and then render the output.

Graphing: A Simple Example

Suppose you want to produce a vertical bar chart that depicts average yearly rainfall worldwide, by month. You can first create a graph template in Illustrator, and then use Graphics Server to apply the data and render the graph.

The following instructions describe how to produce a simple, graph to depict rainfall:

- 1 Start Illustrator, and create a new document.
- 2 In the Graph palette, select the Column Graph tool.



- 3 Draw a column graph in the new document.

6 You can then use Illustrator to customize the look of the graph. For example, you can apply patterns, colors, or illustrations to the bars.



7 Save the document as SVG.

8 At this point, you've created a graph template that Graphics Server can use. As the final steps, we need to write code that dynamically replaces the graph data with new values, and then render the output as a JPEG file. In the following example, we've used a set of class libraries called Exogen.Graphics Server. These libraries greatly simplify the task of directly interacting with the Graphics Server server by encapsulating the process of running a series of commands and dealing with the resulting output. They also add functionality for manipulating XML data sets that are applied to a template.

```
Graphics ServerJob job = new Graphics ServerJob();
LoadContent load_content = new
LoadContent("file:///htdocs/mediaApp/graphing/example_graph.svg");

LoadData load_data = new LoadData();
load_data.setData(xml_data);
SaveOptimized save_optimized = new SaveOptimized("graph.jpg");

job.addOperation(load_data);
job.addOperation(load_content);
job.addOperation(save_optimized);

ResultVector results = job.getResults();
results.writeResultsToDisk("/htdocs/mediaApp/graphing/");
```

Variables Make Graphs Dynamic

If we want to change the data values for our graph, we can use the Graphics Server `<loadContent...>` command to load an XML description of the new values. This method is similar to replacing text and image variables in an SVG document. If you recall, when we made the graph object a variable in Illustrator, the name for the object became “*Variable1*.” We can specify a new value for this variable just like we can specify a new value for a text or image variable, although a few extensions differentiate graph variables from other types of variable data. The XML description of the values to be graphed are specified by a group of elements that are children of the actual variable (that is, *Variable1*) that is assigned to this dynamic graph in the Illustrator Variables palette.

You can see in the following XML command file that the `<Variable1>` element has a child element called `<data>`. This element has a property named *numDataColumns*, which you can use to limit the number of data columns in a graph. In this case, it is set to 12, which is the same number of columns in the original graph. You can change this number, and Graphics Server adjusts the graph scale to accommodate the new number of data points.

The `<data>` element has two child elements, `<propertyRow>` and `<values>`. The `<propertyRow>` element contains the names of the columns being graphed (in our example, the months of the year). The `<values>` element contains one or more sets of `<row>` elements. These `<row>` elements contain the actual data values being graphed (in our example, inches of rainfall). If multiple `<row>` elements are present, then the graph displays multiple data sets in a way that is appropriate to the graph type (that is, side-by-side bar graphs or multiple pie graphs).

Note: You must include an empty value for the first child `<propertyRow>`. This empty value reserves a place for a corresponding `<value key=name></value>` element name of a row title. You also need to set the *numDataColumns* property to equal one more than the number of data columns. Failure to do so may cause data to appear incorrectly.

Applying Data to the Template

After you load the data into the XML commands, all you need to do is apply the variables to an SVG template. To use the implicit `<applyVariables...>` command of Graphics Server, include a `<loadContent out="data">` command followed by a `<loadContent.../>` command that references the SVG template. Graphics Server automatically applies the variables to that template. You can then save the content in Graphics Server to render the graph in the appropriate format.

The actual XML command processed by Graphics Server might look like this:

```
<loadContent out="data">
  <data>
    <Variable1>
      <data numDataColumns="12">
        <propertyRow key="name">
          <value></value>
          <value>January</value>
          <value>February</value>
          <value>March</value>
          <value>April</value>
          <value>May</value>
          <value>June</value>
          <value>July</value>
          <value>August</value>
          <value>September</value>
          <value>October</value>
          <value>November</value>
          <value>December</value>
        </propertyRow>
        <values>
          <row>
            <value>10</value>
            <value>20</value>
            <value>30</value>
            <value>40</value>
            <value>40</value>
            <value>80</value>
            <value>50</value>
            <value>30</value>
            <value>30</value>
            <value>45</value>
            <value>23</value>
            <value>86</value>
          </row>
        </values>
      </data>
    </Variable1>
  </data>
</loadContent>
<loadContent source="file:///lib/graph1.svg" />
<saveOptimized name="graph.jpg" appendExtension="false" />
```

The resulting graph looks like this, rendered as a JPEG file:

Fill in the data values for each month, then click the "Update Graph" button to draw the updated graph

	1998	1999	2000
\$125000	\$250000	\$375000	\$5000000
2001	2002	2003	
\$625000	\$750000	\$875000	\$1000000

Update Graph



XML Graph Display Commands

By directly manipulating the underlying XML, you can change the look and feel of graphs. Specifically, a `<format>` element contains the formatting description for the chart. It has many properties and child elements that describe every aspect of a graph's appearance.

Valid child elements of the `<format>` element include the following:

- `<legend>`
- `<majorAxis>`
- `<minorAxis>`
- `<ignoredDataColumns>`
- `<axis>`
- `<graphDesc>`
- `<xforms>`

The `<legend>` element is optional and doesn't have any child elements. It is used to specify the legend for the graph. The `<majorAxis>` and `<minorAxis>` elements are required, and you can include only one of each. They describe the direction and orientation of the axis, and they each have a single `<dataColumns>` element, which describes the columns of data that can be graphed on each axis. The `<majorAxis>` and `<minorAxis>` elements describe how the columns of graph data may be used within a graph. In addition, an `<axis>` element describes the axis that will be drawn. It differs from the `<majorAxis>` and `<minorAxis>` elements in that it describes how the axis actually appears, including ticks, spines, and labels. The `<graphDesc>` element can be used to create multiple graphs of different types using the same data or a subset of the data. Finally, the `<xforms>` element and its child elements describe transformations that are applied to the graph. These transformations can include changes in the stacking order, text attributes, visibility, and other properties.

Graphing: A Second Example

In this second example, we illustrate how to modify the XML code in an SVG template to change the look and feel of a graph. This technique can be incredibly powerful and is accomplished by modifying the `<format>` element and its children. Depending on the application, you might want to selectively change only a few elements within the `<format>` element. Or perhaps you might want to replace the entire `<format>` element. In either case, you can then use Graphics Server to apply new data values to the graph and render it in the format you desire.

We start by modifying the graph in the first example, changing the graphing axis from vertical to horizontal. Open the SVG template in a text editor, and search for the `<switch...>` statement, similar to this:

```
<switch id="XMLID_1_" i:objectNS="&ns_graphs;" i:objectType="graph" font-family="Myriad-Roman" transform="matrix(1 0 0 1 17.2305 215.6812)">
```

This line is the start of the graph foreign object. If you look a little farther, you will see a section similar to this:

```
<format type="2D" width="356.5" height="380.75" axisLayout="cartesian"
  seriesStackingOrder="firstInFront" inSeriesStackingOrder="lastInFront">
<legend position="right" order="reversed"></legend>
<majorAxis type="enumeration" direction="y" orientation="+">
  <dataColumns>
    <int>0</int>
  </dataColumns>
</majorAxis>
<minorAxis type="value" direction="x" orientation="+">
  <dataColumns>
    <other></other>
  </dataColumns>
</minorAxis>

  <axis name="labels" type="major" ticksAndLabels="explicit"
    positionBase="minEdge" labelPositionBase="1" labelSpacing="1"
    axisVisibility="none" tickLength="none">
<labelString>%m:name</labelString>
</axis>

  <axis name="category" type="major" ticksAndLabels="explicit"
    positionBase="dataAbsMin" tickLength="short" subTickLength="short"
    tickPositionBase="1.5" tickSpacing="1" subTicks="1">
</axis>

  <axis name="value" type="minor" ticksAndLabels="aiDefaultValue"
    positionBase="minEdge" tickLength="short" subTickLength="short"
    subTicks="1">
<labelString>%v inches</labelString>
</axis>
```

This is the `<format>` element for the graph we created in the first example. We'll modify some of these elements and add some child elements to the `<xforms>` element to change the appearance of the graph. For clarity, we'll make these changes manually. Keep in mind that most applications require additional programming to make this process truly dynamic.

Some interesting changes we could make to our graph could be rotating the axis by 90 degrees, or reversing the order of the legend so that it matches the order of the columns. While these changes are relatively minor, they serve well as examples of other changes you might want to make.

Next, open the SVG template in a text or an XML editor. Look for a `<switch.../>` element that begins with

```
<switch id="XMLID_1_" i:objectNS="&ns_graphs;" i:objectType="graph"...
```

This element is the beginning of the description for the graph object. The first child element should be a `<foreignObject>`. This element has a child named `<graph>` that contains the `<data>` and `<format>` element groups. For this example, we want to change some of the values in the `<format>` element.

The first child element we will change is the `<legend>` element. Change the order property from “normal” to “reverse.” This change reverses the order of the legend, making it match the order of the columns. Next, rotate the direction of the graph’s axis so that it becomes a horizontal bar chart instead of a vertical one. Find the `<majorAxis>` element, and change the value of the property “direction” from “x” to “y.” Then, find the `<minorAxis>` element, and change its “direction” property from “y” to “x.”

Suppose you also want to change the label on the graph’s units from centimeters to inches. Because the units are on the minor axis of the graph, you need to find the `<axis>` element whose type property is “minor.” The `<axis>` element has a child element named `<labelString>`. Its value is “%v” centimeters. The “%v” is a placeholder for the value of the data element, and the centimeter is the current label that is placed after the value. Changing centimeters to inches causes the ticks on the axis to be labeled in inches.

You can apply many other transformations to an SVG graph object, including complex operations that can change the symbols used for each column, or that can graph the same data or subsets of the data in multiple ways in a single graph object. Making these changes by hand is an excellent way to develop a template, but to reach the full potential of Graphics Server, we recommend that you use an XML parsing library.

After you finish, you can apply the techniques from the first example to allow you to render the graph with multiple data sets:



Automation Advantage

As you can see, Graphics Server can automate your graph workflow—easing design resources and reducing costs because changes are made programmatically and not manually. In addition, Graphics Server may open new opportunities for including visual presentations of data in company documents. The possibilities are limitless for you to convey information more readily with graphs created with Graphics Server.