Multigraph Users' Guide

version 3.1rc3

Mark Phillips

Introduction

Multigraph is a software system that allows you to create 2-dimensional scientific data graphs in web pages. It allows you to customize the appearance of the graph to your liking very easily, and can plot data from a variety of formats. Multigraph graphs are interactive --- they allow the user to change the scale(s) on the horizontal and/or vertical axes in the graph on the fly. Multigraph also has the ability to read data from a web service, which allows it to be used to "surf" through large datasets, downloading only those the parts of the data that are needed for display.

A graph is created in Multigraph by writing an XML file that describes the details of the graph, including which axes to draw, how to label them, where to find the data to be plotted, and the styles and colors to be used in plotting the data. No programming is required; to publish a web page containing a Multigraph graph, you simply create the XML file describing the graph, and an HTML file that invokes Multigrah with that XML file, and place both of these files on a web server.

Multigraph uses the Adobe Flash browser plugin, which is supported (and usually installed) in almost all desktop web browsers. In particular, it works in Firefox, Internet Explorer, and Safari. Knowledge of Flash programming is not required for creating or using Multigraph graphs, however.

For those who are programmers, though, Multigraph is also available as an Adobe Flex library component, so that it may be used in other Flex applications.

Multigraph is distributed free of charge under the terms of the RENCI Open Source Software License v. 1.0. For details, see Chapter 10, *License*.

Chapter 1. Obtaining and Installing Multigraph

To get started creating graphs with Mulitgraph, download the file main distribution zip file from http://www.multigraph.org/download and unpack it. The zip files contains two things: a file named "Multigraph.js", and a folder named "Multigraph". Put both of these somewhere on your web server; they should both be in the same folder. If you just want to create graphs using local files that you view on your computer, without using a web server, you can put "Multigraph.js" and "Multigraph" in a folder on your computer.

Creating a graph with Multigraph involves writing two files: an XML file that describes the graph, and an HTML file that displays the graph. The next section of this user guide, Chapter 2, *Quick Start*, will walk you through a simple example of each of these files. In the simplest situation, the Multigraph installation (i.e. the "Multigraph.js" file and "Multigraph" folder) can be in the same folder with the XML file and HTML file. In general, however, a single Multigraph installation can be used to support many HTML files that display a variety of graphs described in different XML files; you do not need a separate copy of the Multigraph installation for each graph. In this situation, it makes sense to install the "Multigraph.js" file and the "Multigraph" folder in some centralized place on your web server, and write your HTML files to reference Multigraph.js in that location.

If you are a programmer interested in including Multigraph in your own custom Flex application, see Chapter 7, *Using the Multigraph Flex Component*.

Chapter 2. Quick Start

Before working through the steps described here, download and install a copy of Multigraph as described in the previous section, Chapter 1, *Obtaining and Installing Multigraph*. This is a simple process of unpacking a zip file and copying one file and one folder from it to your computer or web server.

There are two steps involved in creating a web graph using Multigraph. The first is to create a web page (HTML file) that loads the javascript file "Multigraph.js". Somewhere in the HTML page, create a "div" tag and assign an "id" attribute to that div tag. The page body's "onload" function should create a new instance of a "Multigraph" object, giving the div's id attribute value, the name of a "mugl" file describing the graph (see below), and the desired width and height of the graph in pixels. Example 2.1, “Simple Example Multigraph HTML File (graph.html)” shows a simple example.

Example 2.1. Simple Example Multigraph HTML File (graph.html)

```
<html>
  <head>
    <script src="Multigraph.js"></script>
  </head>
  <body onload="new Multigraph('graphdiv', 'graph.xml', [400,300]);">
    <div id="graphdiv"/>
  </body>
</html>
```

In this example, we assume that the "Multigraph.js" file and "Multigraph" folder are in the same folder where you create the HTML file. If you have installed them somewhere else, adjust the `src="Multigraph.js"` line accordingly.

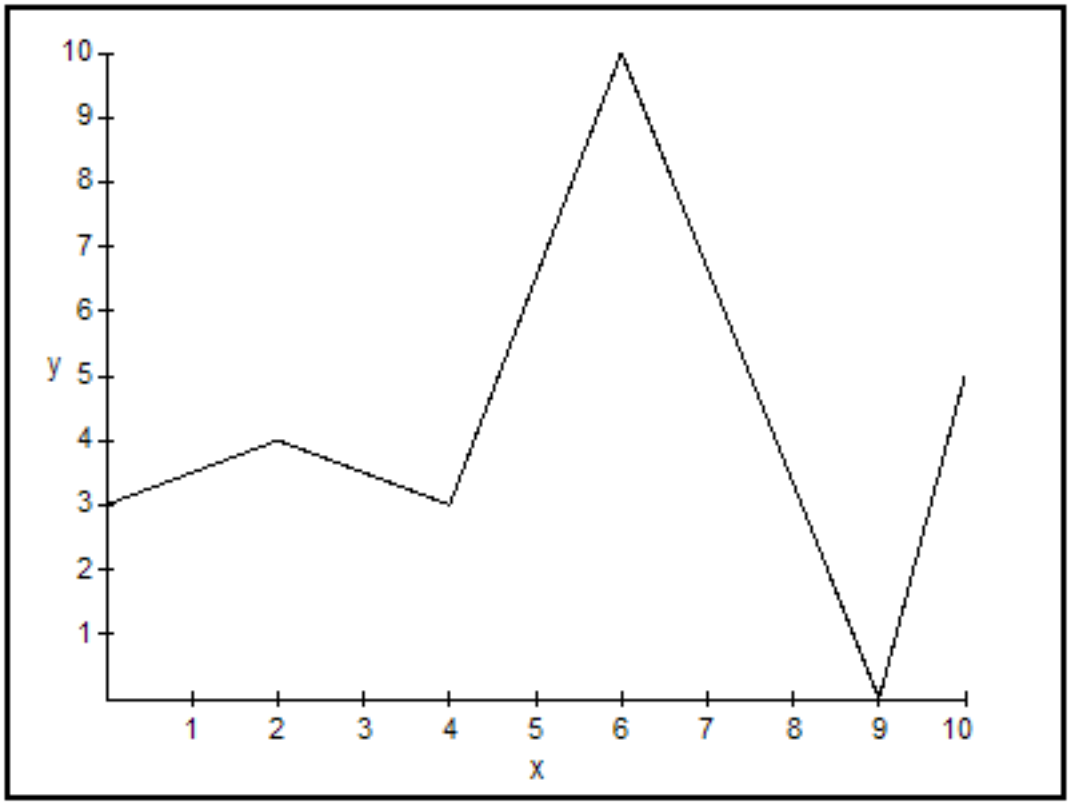
The second step involved in creating a graph consists of writing a "mugl" (pronounced "muggle") file. "Mugl" is short for "Multigraph XML" and is a type of XML file that specifies the data to be graphed, and any preferences you have about how to display the data, axes, and other elements of the graph. The mugl file associated with the above example HTML file is called "graph.xml" (since that name is mentioned in the "new Multigraph(...)" line); Example 2.2, “Simple Example Mugl File (graph.xml)” shows a minimal example of such a file.

Example 2.2. Simple Example Mugl File (graph.xml)

```
<?xml version="1.0"?>
<mugl>
  <data>
    <values>
      0, 3
      2, 4
      4, 3
      6, 10
      9, 0
      10,5
    </values>
  </data>
</mugl>
```

This very small mugl file indicates that the graph should show a plot of the six points whose coordinates are specified in the body of the `values` element. You can try putting this "graph.html" file and "graph.xml" file in the folder where you've installed "Multigraph.js" and the "Multigraph" folder, and then try loading "graph.xml" in your web browser. You should see a graph that looks like Figure 2.1, “Simple Example Graph”.

Figure 2.1. Simple Example Graph



This simple example just scratches the surface of what is possible with multigraph. In this example, which does not specify anything about the style of the graph or the axes or their labels, multigraph makes a lot of default assumptions about how to create the graph, such as choosing a black "line" style graph, and horizontal and vertical ranges to match the given data. In general, these things and many more can be explicitly specified in the mugl file, which allows everything about the graph to be customized.

Chapter 4, *Mugl Files* describes in detail how to write mugl files.

Chapter 3. Creating Multigraph Web Pages

The first part of any web page containing a multigraph graph is the HTML file for the web page. As described in Chapter 2, *Quick Start*, the HTML file should load the javascript file "Multigraph.js". This should be done in a `script` tag in the `head` part of the page.

There are two other things that need to happen in the HTML file. The first is to create a `div` element somewhere in the page, and assign some value to its `id` attribute. This `div` is where multigraph will draw the graph, and it can be anywhere in the HTML page. The second is to write a line of the form

```
new Multigraph(DIV_ID, MUGL_FILE, [WIDTH,HEIGHT]);
```

where DIV_ID is the value that you chose for the `div` containing the graph, MUGL_FILE is the name of the a mugl file describing the graph, and WIDTH and HEIGHT are the width and height that you want (in pixels) for the graph. Typically this line should go in the page body's `onload` attribute, so that it is called only once, when the page is initially loaded.

A web page may include more than one Multigraph graph. Just create a separate `div` for each graph, each with its own unique `id`, write separate calls to the Multigraph constructor for each one, and create separate mugl files for each one. If your page contains more than one graph, you may want to put these Multigraph constructor calls into a function, and then call that function in the page body's `onload` attribute, in order to keep the `onload` value simple.

To view your graph, just load the HTML page into a supported browser (Firefox or Safari). When you publish the page to a web server, put your graph's mugl file(s) in the same folder or directory on the server where you put the HTML file. (Unless your HTML file refers to it with a relative path, in which case you should put it in the correct relative location.)

Chapter 4. Mugl Files

4.1. General Structure

"Mugl" files are the way that you specify to multigraph what data should be plotted, how it should be plotted, and any other settings that determine the way the graph looks. Mugl (pronounced "muggle") is short for "Multigraph XML", and is a type of XML file. Each mugl file should begin with the standard `<?xml version="1.0"?>` declaration, followed by the `<mugl>` tag. The general structure of a mugl file is shown in Example 4.1, “General Structure of a Mugl File”.

Example 4.1. General Structure of a Mugl File

```
<?xml version="1.0"?>
<mugl>
  <window .../>
  <plotarea .../>
  <horizontalaxis .../>
  <verticalaxis .../>
  <plot ...>
    ...
  </plot>
  <data ...>
    ...
  </data>
</mugl>
```

The `window` element gives the size of the graph, the size of the border to be drawn around it, and the margin amounts inside the border. There can be at most one `window` element; if it is omitted, multigraph chooses default values for the border and margins, and relies on the window size given in the Multigraph.Graph constructor.

The `plotarea` element indicates the region inside the graph window where the data will be plotted. There can be at most one `plotarea` element; if it is omitted, multigraph chooses a reasonable default.

The `horizontalaxis` and `verticalaxis` elements give properties related to the display and function of the graph's axes. There can be at most one `horizontalaxis` element, but any number of `verticalaxis` elements are allowed.

The `plot` element indicates properties of the data plot itself, such as style, color, which data variables to plot, and which axes to plot them against. There can be any number of `plot` elements.

The `data` element gives the data to be plotted, either by providing the data in a `data` subelement, or by giving the location of a file or web service from which the data can be read.

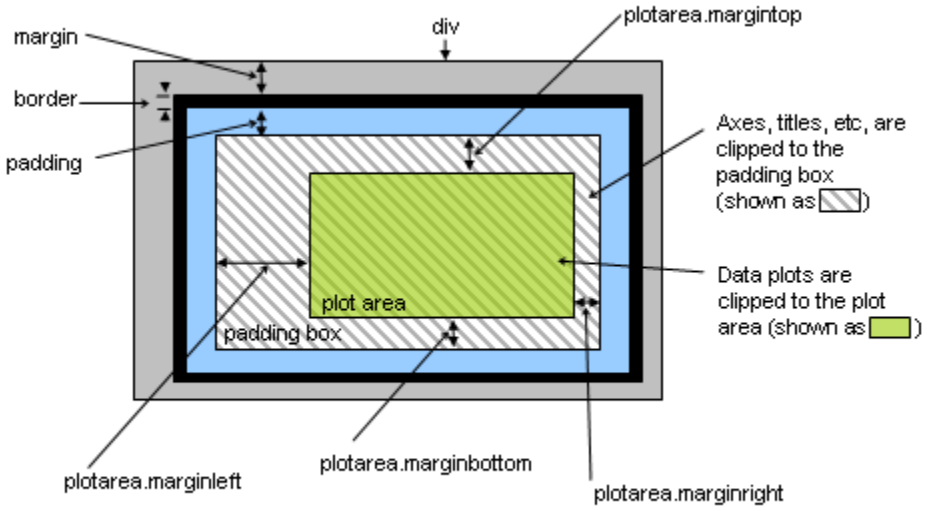
The following sections describe these elements in more detail.

4.2. Borders, Margins, and the Plot Area

Multigraph draws each graph into a `div` in the HTML page. There are several things that you can control about the way multigraph lays out the graph in that `div`. In particular, multigraph can optionally draw a rectangular border around the edge of the `div`, possibly with some space inside of and/or outside of that border. You can also control the part of the `div` where data is drawn, and where axes, titles, and other elements of the graph are drawn.

All these things can be described in terms of a series of (mostly invisible) nested rectangles inside the graph's `div` which are shown in Figure 4.1, “Border, Margin, Padding, and the Plot Area”.

Figure 4.1. Border, Margin, Padding, and the Plot Area



The outermost rectangle in this figure --- the one shown in solid gray --- corresponds to the `div` itself. Inside that rectangle, inset by an amount called the "margin" from all four sides, is a black frame called the "border". Inside the border is another rectangle called the "padding box", inset from the border by an amount called the "padding" amount (shown in blue in the figure) on all four sides. The values of these inset amounts, as well as the thickness of the border, and optionally the overall size of the `div`, are all specified with the `window` element in the mugl file. It accepts the following attributes, all of which are optional:

`width` The width, in pixels, of the graph. Multigraph assigns this width to the `div` element in the HTML page containing the graph. A width specified in the Multigraph.Graph constructor call takes precedence over this value, and in that case, this attribute value may be omitted. is

`height` The height, in pixels, of the graph. Multigraph assigns this height to the `div` element in the HTML page containing the graph. A height specified in the Multigraph.Graph constructor call takes precedence over this value, and in that case, this attribute value may be omitted.

`margin` A number of pixels of blank space to allow between the window and the border, on all 4 sides. If this attribute is omitted, it defaults to 2.

`border` The width, in pixels, of a border to draw around the graph. If this attribute is omitted, it defaults to 2.

`padding` A number of pixels of blank space to allow inside the border; everything else (other than the border) that multigraph draws will be clipped to an imaginary rectangle that is inset by this number of pixels from all 4 sides of the border. If this attribute is omitted, it defaults to 5.

There can be at most one `window` element in a mugl file. If it is absent, all of the above attributes assume their default values.

Note that in Figure 4.1, “Border, Margin, Padding, and the Plot Area”, the margin, border, and padding are shown in different colors (the margin is gray, the border black, and the padding is blue), and are all much larger than they normally would be. In a real multigraph graph, the margin and padding are white, the border is black, and they are all much smaller. You can explicitly set any of them to 0 if you want to eliminate it altogether.

Inside the padding box is another rectangle called the "plot area"; it is inset from the padding box by four (possibly) different amounts along each edge, called the "plot area margins". The plot area is the region where multigraph plots data --- all data plots are clipped to this rectangle. The plot area is shown in green in the figure above, but in a real graph it is invisible.

In general, multigraph uses two rectangles for clipping: the padding box and the plot area. The padding box (which includes the plot area) serves as the clipping region for the whole drawing --- with the exception of the border, everything that

multigraph draws, including axes, labels, and titles, is clipped to the padding area. In other words, the padding area defines the region of the `div` in which multigraph does its drawing. The plot area is the subset of the padding box where the data itself is plotted.

The `plotarea` element in the mugl file defines the plot area. It accepts the following attributes, which represent offsets relative to the padding area:

`marginbottom` Offset from the bottom side. Default: 30.
`marginleft` Offset from the left side. Default: 30.
`margintop` Offset from the top side. Default: 10.
`marginright` Offset from the right side. Default: 30.

The `plotarea` element also accepts the following attributes which control an optional border to be drawn around it:

`border` The width, in pixels, of a border to be drawn around the plotarea. The default is 0, which means that no border is drawn.
`bordercolor` The color to use for the plotarea border. The default is "0xeeeeee", which is very light gray, and matches the default value for axis grid lines.

There can be at most one `plotarea` element in a mugl file. If it is absent, all attributes assume their default values.

4.3. Axes

A Multigraph graph has one or more horizontal axes, and one or more vertical axes. The location and other display properties of the axes are specified with the `horizontalaxis` and `verticalaxis` elements in the mugl file. This includes mathematical information such as the data type and range of values along the axis, as well as display information such as how the axis should be labeled, a title to be drawn for the axis, and so on.

A mugl file can contain any number (0 or more) of `horizontalaxis` and `verticalaxis` elements. Each graph always contains at least one horizontal axis and at least one vertical axis. If the `horizontalaxis` element is missing, multigraph creates a horizontal axis with all the default settings. The same is true for the the `verticalaxis` tag.

The `horizontalaxis` and `verticalaxis` elements have the following attributes:

`id` An identifier that can be used to refer to this axis elsewhere in the mugl file. If this attribute is missing, it defaults to "x" for the first horizontal axis, and to "x1", "x2", ... for any additional horizontal axes. The default is "y" for the first vertical axis, and "y1", "y2", ... for any additional vertical axes.

`type` Indicates the data type for the axis. Should be one of "number" or "datetime". The default is "number".

`length` The length of the axis. There are two ways to specify the axis length. The first way is to simply give a number between 0 and 1, which represents a fraction of the parallel dimension of the plot area (width for horizontal axes, height for vertical axes). So, for example, for a horizontal axis, a value of "1" means that the axis length is exactly the width of the plot area, and a value of "0.5" would mean that the axis is half as long as the width of the plot area.

The second way to specify an axis length is to give an expression of the form "A+B" or "A-B". In this case, the first number A represents a fraction of the parallel dimension of the plot area, as above, and the second number +B or -B represents a number of pixels to add or subtract from that length. So, for example, a length of "1-10" for a horizontal axis would mean 10 pixels less than the width of the plot area. A value of "0.5+20" for a vertical axis would mean 20 pixels longer than half the height of the plot area. If for some reason you want to specify an axis length exactly in terms of pixels, use 0 for A; so for example, "0+200" would give an axis that is 200 pixels long. (Note that you definitely need to specify the "0+" in this case; if you specify a length simply as "200", Multigraph will interpret it, according to the first method above, as 200 times the width or height of the plot area, which is almost certainly not what you want.)

`anchor`

`base`

`position` These attributes give the location of the axis relative to the plot area, according to the conventions described in Section 4.6.1, “Positioning”. `base` is the location of the base point relative to the plot area; the default value for `base` is "-1 -1" --- i.e. the lower left corner of the plot area. `anchor` is a single number (not a coordinate pair) between -1 and 1 indicating the location of the anchor point along the axis. The default is "-1", which represents the left endpoint of a horizontal axis, or the bottom endpoint of a vertical axis. `position` is a coordinate pair of pixel offsets, as described in Section 4.6.1, “Positioning”; its default value is "0 0".

Versions of Multigraph prior to version 2.3 used a slightly different axis positioning system, in which the `position` attribute was a single number indicating a pixel displacement from the bottom (for horizontal axes) or left (for vertical axes) edge of the plot area. To provide backward compatibility, if `position` is specified as a single number, the current version of Multigraph will interpret it to be a displacement in the direction perpendicular to the axis; this behavior, together with the default values of the `base` and `anchor` attributes (which did not exist in the older versions of Multigraph), gives backward compatibility with the older style of axis positioning. In new MUGL files, you can often achive the desired axis positioning using only the `length`, `base`, and/or `anchor` attributes, and accept the default value of "0 0" for the `position` attribute.

`positionbase` Deprecated. The current version of Multigraph accepts this option to provide backward compatibility with older versions, but it should not be used in new MUGL files. Use the `base` attribute instead to switch a vertical axis to be placed relative to the right edge of the plot area (base="1 -1", for example), or to switch a horizontal axis to be placed relative to the top edge of the plot area (base="-1 1", for example).

`pregap`

`postgap` Deprecated. Do not use these.

`min` The minimum data value for the axis --- i.e., the data value corresponding to the left (for horizontal axes) or bottom (for vertical axes) endpoint of the axis. If the axis is a "number" type axis, this value is a number. If it is a "datetime" type axis, the value is a datetime in the format YYYYMMDDHHmmss.

Alternatively "min" value may be the keyword "auto", which indicates that the minimum data value for the axis should be computed from the given data. This only works for data sources where the entire data set is available at the outset --- it does not work for web-service based data sources.

The default value for the "min" attribute is "auto".

`minposition` An expression that indicates the location along the axis corresponding to the minimum data value (before any panning or zooming that the user might do). This is expressed in general in the form "A+B" or "A-B", where A is a relative coordinate between -1 and 1, and B is pixel offset. The "+B" or "-B" is optional; if it is not present, B is taken to be 0. So, for example, minposition="-1" corresponds to the left or bottom endpoint of the axis; this is the default. minposition="-1+5" corresponds to a point that is 5 pixels to the right of the left endpoint, or above the bottom endpoint.

`max` The maximum data value for the axis --- i.e., the data value corresponding to the right (for horizontal axes) or top (for vertical axes) endpoint of the axis. If the axis is a "number" type axis, this value is a number. If it is a "datetime" type axis, the value is a datetime in the format YYYYMMDDHHmmss.

Alternatively "max" value may be the keyword "auto", which indicates that the maximum data value for the axis should be computed from the given data. This only works for data sources where the entire data set is available at the outset --- it does not swork for web-service based data sources.

The default value for the "max" attribute is "auto".

`maxposition` An expression that indicates the location along the axis corresponding to the maximum data value (before any panning or zooming that the user might do). This is expressed in general in the form "A+B" or "A-B", where A is a relative coordinate between -1 and 1, and B is pixel offset. The "+B" or "-B" is optional; if it is not present, B is taken to be 0. So, for example, maxposition="1" corresponds to the right or top endpoint of the axis; this is the default. maxposition="1-5" corresponds to a point that is 5 pixels to the left of the right endpoint, or below the top endpoint.

Note that is is possible to specify a `minposition` and `maxposition` combination in which the min position is to the right of, or above, the max position, which will cause the data values along the axis to increase in the opposite direction from usual (increasing leftward for horizontal axes, or downward for vertical axes). This is perfectly valid and will work correctly, but you should make sure that it is really what you want before deciding to make use of this feature, because it is likely to be confusing to users, unless there is a good reason to portray your data in this manner.

Figure 4.2, “Horizontal Axis Specifications” and Figure 4.3, “Vertical Axis Specifications” show how these attribute values determine the way that each kind of axis is displayed relative to the plot area. Note that the position of each axis is relative to an edge of the plot area, but all or part of the axis may lie outside the plot area (in the padding box). In particular, a negative `position` value represents a distance away from the plot area (into the padding box), and negative `pregap` and `postgap` values cause the ends of the axis to extend into the padding box.

Figure 4.2. Horizontal Axis Specifications

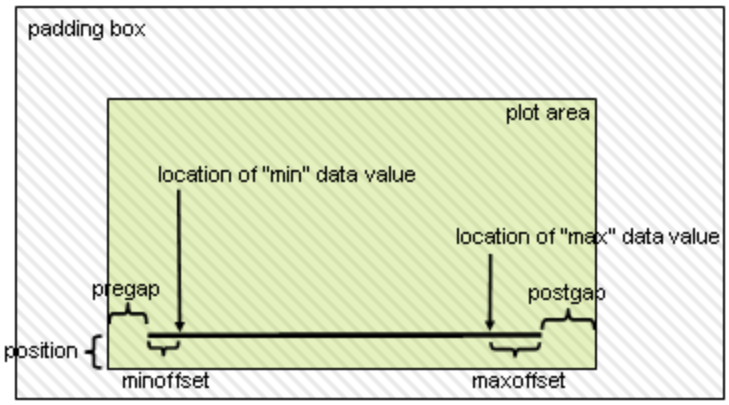
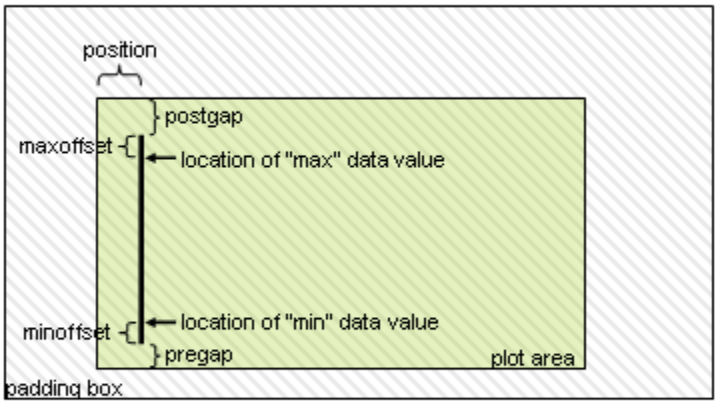


Figure 4.3. Vertical Axis Specifications



In addition to the above attributes, the `horizontalaxis` and `verticalaxis` elements may contain the following subelements:

`title` The text of this element is used as a title for the axis; it is positioned and oriented according to the values of the `position`, `anchor`, and `angle` attributes described in Section 4.6.1, “Positioning”. The title is positioned relative to the the center of the axis; the "position" is a pixel offset from this point. (The "base" point mentioned in Section 4.6.1, “Positioning” The default anchor point depends on the choice and position of the axis. For a horizontal axis, the default anchor point is "0 1" if the axis is positioned at the bottom of the graph (positionbase="bottom"), or "0 -1" if the axis is positioned at the top of the graph (positionbase="top"). For a vertical axis, the title's default anchor point is "1 0" if the axis is positioned to the left of the graph (positionbase="left"), or "-1 0" if the axis is positioned to the right of the graph (positionbase="right"). The default angle is always 0. The default title position also depends on the axis, and has been chosen to look reasonable in most common situations; see the source file Config.as if you want to know exactly what the default position values are.

If there is no `title` element, the axis's title defaults to its `id`. To prevent a title from being drawn at all, use an empty `title` tag ("`<title/>`").

`labels` The `labels` element specifies how multigraph will draw tick marks along the axis, and text labels for the tick marks. It has two possible forms, depending on whether it has a "spacing" attribute. In the first case, when it does have a "spacing" attribute, it takes the following attributes:

`format` A string describing the format to be used for tick labels. The syntax for this string depends on the type of the axis. For 'number' type axes, it should be a C-style "printf" format string, such as '%1d', which is the default. For 'datetime' type axes, it should be a string containing some combination of the following special format characters, each of which should be preceeded by a '%':

Table 4.1.

Character	Expansion
Y	four digit year
y	two digit year
d	day of month without leading zero
D	2-digit day of month with leading zero
h	hour of day, 12 hour clock
H	hour of day, 24 hour clock
i	minutes
m	month number without leading zero
M	2-digit month number with leading zero
N	month name, spelled out
n	month name, 3 letter abbreviation
p	AM or PM
p	am or pm
s	seconds
W	weekday name, spelled out
w	weekday name, 3-letter abbreviation
L	newline
%	%

Any characters in the format string that are not preceeded by a '%' are included verbatim in the formatted result.

`start` A data value indicating where tick marks should be aligned on the axis. The location of axis tick marks is determined by the combination of this "start" value and a "spacing" value; multigraph will draw tick marks (and labels for the tick marks) at locations T + n*S, where T is the "start" value, S is the "spacing" value, for each integer n such that T + n*S is visible in the graph's padding area. The default "start" value is 0.

`spacing` A list of values, separated by spaces, giving the possible spacings to be used between tick marks on the axis. These numbers should be sorted in decreasing order. Multigraph will attempt to choose the densest tick spacing possible from amoung these choices that results in labels that do not overlap. The default value for the "spacing" attribute is "10000 5000 2000 1000 500 200 100 50 20 10 5 2 1 0.1 0.01 0.001".

`angle`

`position`

`anchor` These determine the location and orientation of each tick mark's label relative to the location of the tic mark itself. See Section 4.6.2, “Axis Label Positioning” and Section 4.6.1, “Positioning” for further details.

The second possible form for the `labels` element accepts all of the above attributes except the `spacing` attribute; this form takes a sequence of `label` subelements instead. Each label subelement indicates one possible spacing for axis tick marks, along with other settings to be used when that spacing is chosen. Specifically, the `label` subelement accepts the following attributes:

- `format`

- start
- spacing
- angle
- position
- anchor

In the context of a label element, these attributes have the same meaning and form as for the labels element, except that the "spacing" attribute should be a single data value rather than a list of values. The values specified by the attributes of a label element apply only when the spacing for that particular element is in effect. The default value for each of these attributes (except for the "spacing" attribute, which is required), is whatever value would be in effect for the containing labels element.

grid The presence of this element causes Multigraph to draw grid lines perpendicular to this axis, at the location of each tick mark. The grid element takes one optional attribute:

color A hexadecimal number giving the color to be used for the grid lines. The default is "0xeeeeee", which is very light gray.

If a horizontalaxis or verticalaxis element does not contain a grid subelement, Multigraph does not draw grid lines along that axis. Note that the grid element only affects the drawing of grid lines associated with the axis it appears in. If you want both horizontal and vertical grid lines, be sure to include a grid element in both the horizontalaxis and verticalaxis sections of the mugl file.

pan This element can be used to configure the type of panning that multigraph allows the user to do for this axis. It accepts the following attributes:

allowed One of the words "yes" or "no"; default is "yes".

min The minimum data value to ever be displayed for this axis; panning is never allowed to go below this number.

max The maximum data value to ever be displayed for this axis; panning is never allowed to go above this number.

zoom This element can be used to configure the type of zooming that multigraph allows the user to do for this axis. It accepts the following attributes:

allowed One of the words "yes" or "no"; default is "yes".

min This attribute controls how far "in" the user can zoom on this axis; its value is a distance along the axis, and is the smallest interval that the user will be allowed to zoom in to. For a number type axis, this value is just a number, and multigraph will constrain the zooming so that the range of values displayed along the axis is at least that number. For a datetime type axis, the value should be an interval of time consisting of a number followed by one of the letters 'Y', 'M', 'D', 'H', 'm', or 's', which indicate years, months, days, hours, minutes, or seconds, respectively. For example a value of "6M" will cause multigraph to constrain zooming along the axis so that the range of time displayed along the axis is always at least 6 months. If this attribute is missing, the default behavior is for unconstrained inward zooming.

max This attribute controls how far "out" the user can zoom on this axis; its value is a distance along the axis, and is the greatest interval that the user will be allowed to zoom out to. For a number type axis, this value is just a number, and multigraph will constrain the zooming so that the range of values displayed along the axis is at most that number. For a datetime type axis, the value should be an interval of time consisting of a number followed by one of the letters 'Y', 'M', 'D', 'H', 'm', or 's', which indicate years, months, days, hours, minutes, or seconds, respectively. For example a value of "10Y" will cause multigraph to constrain zooming along the axis so that the range of time displayed along the axis is always at most 10 years. If this attribute is missing, the default behavior is for unconstrained outward zooming.

anchor A data value that indicates a location on the axis about which zooming should be centered. Should be either a data value, or the keyword "none". If it is "none", then zooming is centered about the

location of the mouse cursor when the user first presses the mouse button to begin dragging. The default is "none".

`binding` This element can be used to "bind" two or more axes together with a linear mapping, so that interactive panning and/or zooming causes them to move together. It is typically used to connect axes that represent the same vertical scale with different units, such as celsius and Fahrenheit temperature. It can also be used to connect axes in different graphs that represent the same scale (for example, the horizontal time axis). It requires the following three attributes:

- `id` The `id` attribute should be a name that identifies the binding. It can be any string, and all axes having `binding` elements with the same `id` value will be connected to each other.
- `min` A "minimum" value for the binding. Axes bound together in a binding will be connected in such a way that their "min" values correspond to each other.
- `max` A "maximum" value for the binding. Axes bound together in a binding will be connected in such a way that their "max" values correspond to each other.

The `min` and `max` attributes can be any two data values on the axis that determine the linear relationship between an axis and other axes in the same binding. They do not need to correspond to the axis's own min and max values, and in general they will not.

For example, to bind a celsius temperature axis to a Fahrenheit temperature axis, you could use the binding

```
<binding id="tempbinding" min="0" max="100"/>
```

on the celsius axis, and

```
<binding id="tempbinding" min="32" max="212"/>
```

on the Fahrenheit axis.

4.4. Plots

The data that multigraph draws into the plot area is organized into one or more "plots". A plot consists of a selection (usually two) of variables to be plotted, along with information about which axes to plot them along, and the style, color, and other visual attributes of the plot.

A plot is described with the `plot` element in the mugl file. There can be any number (zero or more) of `plot` elements in the file. Multigraph will draw the plots in the order they appear in the mugl file. If no `plot` element is present at all, multigraph creates a single line plot of the data section's first variable on the horizontal axis against the second variable on the vertical axis (or first vertical axis, if there is more than one).

The `plot` element takes no attributes, and the following subelements:

`horizontalaxis` The `horizontalaxis` element indicates which variable should be mapped to the horizontal axis in the plot. It has no attributes, and takes a sequence of `variable` subelements. Each `variable` subelements has a single attribute named `ref` whose value should be the id of a variable from the `data` section. Most plots have just one variable corresponding to the horizontal axis. If a `plot` has no `horizontalaxis` element, it defaults to using the first variable from the `data` along the horizontal axis.

`verticalaxis` The `verticalaxis` subelement indicates which variable(s) should be mapped to the vertical axis in the plot. It has a single attribute named `ref` whose value is the id of one of the `verticalaxis`

elements specified earlier in the file. If this attribute is missing, or if The `verticalaxis` subelement is missing altogether, the plot will use the first vertical axis mentioned in the file (or a default one if none is mentioned). The `verticalaxis` subelement takes a sequence of `variable` subelements. Each `variable` subelement has a single attribute named `ref` whose value should be the id of a variable from the `data` section. Most plots have just one variable corresponding to the vertical axis, but some may have more than one. For example, a plot with error bars would have two "vertical axis" variables: one for the value, and another for the error amount. If a `plot` has no `verticalaxis` element, or a `verticalaxis` with no `variable` subelements, it defaults to using the second variable from the `data` along the vertical axis.

`renderer` The `renderer` subelement specifies the graph style to be used for the plot. "Renderers" are objects in the multigraph software that are responsible for drawing data plots. Each renderer corresponds to a specific plot style, such as points, lines, bars, and so on. The `renderer` subelement has one attribute named `type`, whose value indicates the choice of renderer. New renderers that implement new plot styles can easily be added to multigraph; Section 5.1, “Adding New Renderers” describes how to do this.

Each renderer may have options that can be specified to customize its behavior. Options control things like color, line thickness, and other visual aspects of the plot. Each option is given with a `option` subelement of the `renderer` element. Each `option` takes two attributes: `name` and `value`, which give the name and value for that option.

At the time of this writing the supported renderers are:

Table 4.2.

It is possible that new renderers will have been added to multigraph since this documentation was written. You can generate a list of the current renderers that your copy of multigraph knows about, and the options they support, by writing an HTML file that creates a MultigraphRendererList instance as follows:

Example 4.2. Generating a List of Currently Supported Renderers

```
<html>
  <head>
    <script src="Multigraph.js"></script>
  </head>
  <body onload="new MultigraphRendererList('mydiv');">
    <div id="mydiv"/>
  </body>
</html>
```

To see the list of renderers available for the current copy of multigraph that is available from www.multigraph.org, visit the web page http://www.multigraph.org/renderers.

4.5. Data

The `data` element in the mugl file specifies the data to be plotted, either by listing the data itself in the mugl file, or by describing where multigraph can find the data. The `data` element is the only element that is absolutely required in a mugl file. (Multigraph can make reasonable default decisions about everything else, but it has to be given some data to graph!)

There are three ways that you can specify the data. The simplest is to include the data directly in the mugl file, in a list of commma-separated values. The second way is for the mugl file to refer to data that is contained in a separate comma-separated value (csv) file. The third way is for the data to be provided via a web service, in which case the mugl file indicates the address of that service, and multigraph will use the service to fetch the data as needed.

Regardless of which of the above forms the data is in, it is helpful to think of the data as consisting of a table of values, where each column in the table corresponds to a particular variable, and each row corresponds to a set of values for those variables. For example, a data set containing temperature and precipitation values measured every hour for one day might correspond to a table with 3 columns and 24 rows: one column each for the time of measurement, temperature, and precipitation, and one row for each of the 24 hours in the day.

A mugl file's `data` element should begin with a `variables` subelement that indicates some basic information about the columns in this "table" of data. The `variables` element contains a sequence of `variable` subelements, each of which has the following attributes that give information about a data variable:

`id` An identifier to be used to refer to this variable elsewhere in the mugl file. The default is "x" for the first variable, "y" for the second variable, and "y1", "y2", ... for any additional variables.

`column` A number indicating which "column" the variable is in. The default is 0 for the first variable, 1 for the second, and so on.

`type` The data type of the variable. Should be either 'number' or 'datetime'; the default is 'number'.

After the `variables` subelement in the `data` section, exactly one of the following three subelements should occur:

`values` The `values` subelement should be used when the data is to be included directly in the mugl file. The data should appear between the <values> and </values> tags, with one set of values per line, and individual values on each line separated by commas.

`csv` The `csv` subelement should be used when the data is in a separate comma-separated values file. The `csv` subelement has a single attribute named `location`, whose value should be the name of the csv file.

`service` The `service` subelement indicates that multigraph should fetch data from a REST web service. It has one attriubute named `location` which should give the address of the web service. Because of JavaScript security restrictions, this address must be on the same server where the HTML page containing the graph is. The value of the `location` attribute should be either a root-relative (i.e., starting with '/') path, or a relative path that is relative to the location of the HTML file. In particular, the REST service URL should not include the "http://" prefix or the server name.

4.5.1. Missing Data

Sometimes there is a need to treat certain data values as "missing", which will affect the way that Multigraph plots those values. For example, when drawing a plot that involes lines connecting consecutive data points, Multigraph will leave a gap at locations corresponding to missing data points.

Information about missing data values may be specified in the `variables` subsection of the `data` element. The `variables` tag takes an optional pair of attributes: `missingvalue`, which indicates a value to considered as missing, and `missingop`, which indicates a test to be used to determine whether a given value is missing. The `missingop` attribute should be one of the strings "lt", "le", "eq", "ge", or "gt", which stand for "less than", "less than or equal", "equal", "greater than or equal", and "greater than", respectively. The value of `missingop` determines the operation to be used in determining whether a data value should be considered missing: the data value is compared, using the `missingop` operation, to the `missingvalue` value. For example, if `missingop` is "le" and `missingvalue` is "-9000", then any data values less than or equal to -9000 will be considered missing.

The default for `missingop` is "eq", so the default test is for exact equality.

There is no default for `missingvalue`; if it is not specified, then no data values will be considered missing.

The `variable` tag also accepts the `missingop` and `missingvalue` attributes, in which case they apply only to an individual variable. Values for `missingop` and `missingvalue` given for an individual `variable` element override any that are specified for the enclosing `variables` element. So you can specify `missingop` and `missingvalue` either for each `variable` individually, or globally at the `variables` level, in which case any individual `variable` tags that do not have their own `missingop` and `missingvalue` settings will inherit the values from the inclosing `variables` settings.

4.6. Styling

Although Multigraph constructs much of a graph automatically, some of the most important aspects of a graph cannot be completely determined automatically. These include things like titles, legends, plot styles, color choices, special annotations, and font sizes. A well-designed graph that easily conveys the intended information to its audience will have a carefully chosen combination of these things that fits the data and the context in which it is being presented. This section describes how to control these aspects of a graph.

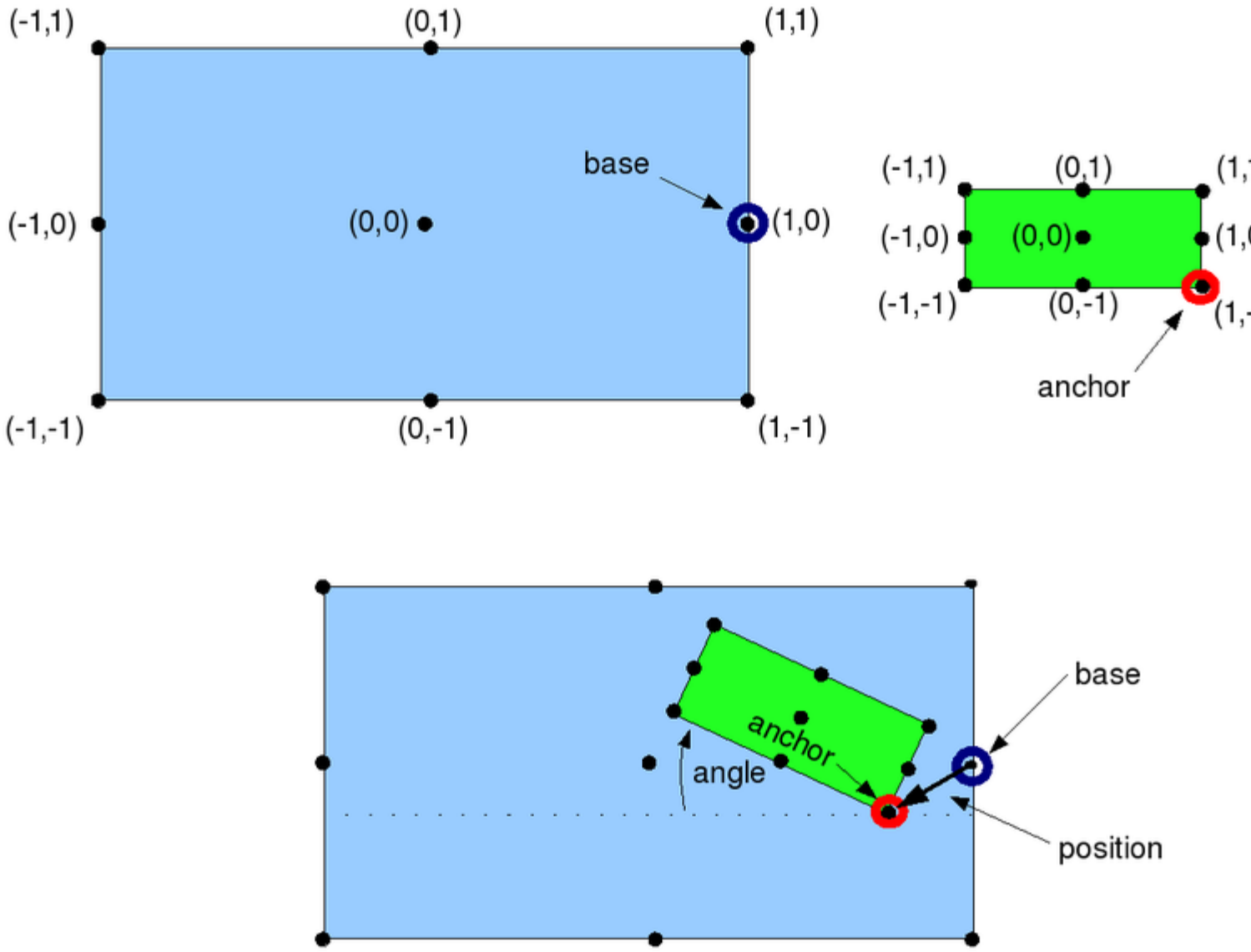
4.6.1. Positioning

Many visual components in Multigraph are positioned relative to something else in the graph. For example, a graph title or legend is typically positioned (centered, left justified, right justified, etc) relative to the entire graph window. A title for an axis is usually positioned relative to the axis, and labels along the axis are positioned relative to the tic marks on the axis. The axes themselves are positioned relative to the plot area. Multigraph uses a common collection of attributes and conventions for relative positioning of various components; this section describes these conventions and attributes.

In the most general case, relative positioning can be thought of as the task of placing a small rectangle, called the "placed rectangle", relative to a large rectangle, called the "base rectangle". (It may help to think in terms of placing the placed rectangle "inside" the base one, but in some cases the placed rectangle will be positioned so that it extends beyond the edges of the base one, or lies outside it completely.) The placed rectangle might represent a legend or a text string, for

example, and the base rectangle represents some rectangular region of the graph window --- for example, the plot area where the data is plotted (see Section 4.2, “Borders, Margins, and the Plot Area”). Each of these rectangles has a width and height in pixels, and locations in the rectangle can be specified by giving pixel coordinates. Since it is sometimes awkward to specify locations in pixels, though, Multigraph also makes use of a "relative" coordinate system for each rectangle, in which the lower left corner has the coordinates (-1,-1), and the upper right corner has coordinates (1,1). The center of the rectangle has coordinate (0,0).

Figure 4.4. Positioning



The top part of Figure 4.4, “Positioning” shows an example in which the base rectangle is blue and the placed rectangle is green, and the "relative" coordinates of several key points in each rectangle are marked. We want to specify a location of the green rectangle relative to the blue one. We do so by choosing a "base" point in the blue rectangle, and an "anchor" point in the green rectangle. We then position the green rectangle in the blue one so that its anchor point lines up with the base point in the blue rectangle, but possibly offset by a vector which is called the "position", and possibly at a given "angle" to the horizontal. The bottom part of Figure 4.4, “Positioning” shows the result.

In Figure 4.4, “Positioning”, the base point is (1,0) and the anchor point is (1,-1). The base point is specified in the relative coordinates of the base rectangle, and the anchor point is in the relative coordinates of the placed rectangle. The "position" is a vector that indicates an offset of the anchor point away from the base point in the final positioning; it is specified in pixels rather than relative coordinates. In the figure, the position vector looks like it might be something like (-30,-20). The "angle" is specified in degrees, with positive angles pointing counterclockwise, so in the figure the angle appears to be roughly -45.

These four quantities, the base, anchor, position, and angle, are available for several tags in mugl files using the `base`, `anchor`, `position`, and `angle` attributes. Values for the `base`, `anchor`, and `position` attributes are specified as a pair of numbers separated by a space, without parentheses. These attributes allow for any arbitrary positioning of the placed rectangle relative to the base rectangle in a way that makes many common alignment possibilities easy to specify. For example, to center a text string (think of the green rectangle as representing a text string) just under the top edge of the blue rectangle, use base="0 1", anchor="0 1", position="0 0", angle="0". If you want to add a few pixels of vertical space between the top of the text and the top edge of the blue rectangle, use something like position="0 -5", which shifts the text down by 5 pixels. To center a text string just ABOVE the top edge of the blue rectangle, use base="0 1", anchor="0 -1", position="0 0", angle="0". To right justify a legend (now think of the green rectangle as a legend) along the middle of the right edge of the blue rectangle, use base="1 0", anchor="1 0", position="0 0", angle="0".

Having the base and anchor points specified in relative coordinates but the position vector in pixels might seem odd at first, but this turns out to be a powerful combintation that allows you to specify locations that do not depend on the exact pixel size of the graph, and that continue to work correctly if and when the size of the graph is changed. Think of `base`, `anchor`, and angle as the main way that an object is positioned, and `position` as an offset to allow for a small amount of padding. Often the amount of padding that looks best depends on the font size being used, or on the pixel size of an object, so `position` is given in pixels. The `base` and `anchor` points, though, specify relative location and alignment and do not depend on specific pixel measurements.

Some mugl tags that use these attributes do not allow all of them; for example, the `legend` tag does now allow the `angle` attribute, because legends cannot be rotated (legends are always displayed with an angle of 0). Most mugl tags that use these attributes have default values for them, so that you only need to specify attributes that differ from the defaults. The default values vary depending on the tag; see the documentation for each tag to find out what its default attribute values are.

In some situations the base point is determined by context and there is no need (or it doesn't make sense) to think in terms of a base rectangle. For example, when positioning an axis title, the base point is the center point of the axis, so the `title` subelement of the axis elements takes `position`, `anchor`, and `angle` attributes that position the title relative to the axis center point, and the `base` attributes is not needed (or allowed).

There is one more attribute that is sometimes used in connection with positioning. Some mugl tags accept the `frame` attribute as a way of identifying the base rectangle. It typically takes values like "plot" or "padding", to indicate that the base rectangle is the plot box or the padding box, respectively; see Section 4.2, “Borders, Margins, and the Plot Area” for a description of these boxes. In most cases, though, the base rectangle is determined by the context and there is no need (or allowance) for a `frame` attribute.

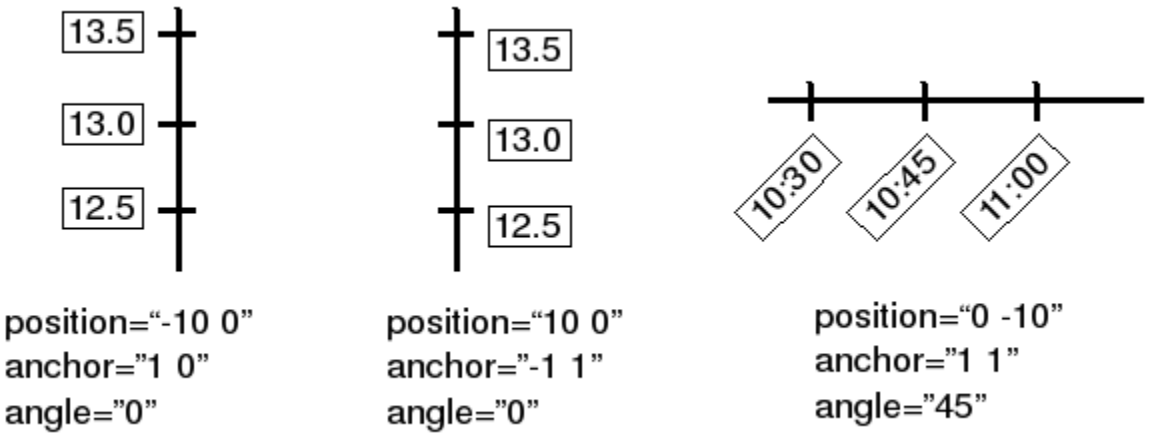
In some cases either the base rectangle or the placed rectangle may actually not be a rectangle, but rather just a line segment. For example, when specifying the location of an axis relative to the plot area, the "placed rectangle" is the axis. In cases such as this, where the placed rectangle is a line (segment), the `anchor` attribute can take a value that is a single number ranging from -1 to 1 that represents a location along the line segment, rather than a pair of numbers. Similarly, in cases where the "base rectangle" is a line segment, such as when positioning an axis title relative to the axis, the `base` attribute can be a single number between -1 and 1.

4.6.2. Axis Label Positioning

The `labels` subelement of the `horizontalaxis` and `verticalaxis` elements allows you to control how Multigraph places tic marks and labels along the axis. The location of the label for each tic mark is determined by the `position`, `anchor`, and `angle` attributes of the `labels` tag. In terms of the discussion in Section 4.6.1, “Positioning”, the "small" rectangle that is being positioned is an imaginary rectangle surrounding the text label, and the "base" point for each label is the location of the corresponding tic mark on the axis (the `labels` tag does not take a `base` attribute).

Through various combinations of the `anchor`, `position`, and `angle` attributes it is possible to position, orient, and justify text in a graph very precisely. Figure 4.5, “Axis Label Placement Examples” shows several examples. Note that the rectangles drawn around the text in the figure are shown just for clarity; in a real graph, Multigraph does not draw these rectangles.

Figure 4.5. Axis Label Placement Examples



4.6.3. Graph Titles

The `title` element allows you to add a title, or caption, to a graph. More documentation for this will be written soon.

4.6.4. Legends

The `legend` element allows you to control the graph's legend. More documentation for this will be written soon.

4.6.5. Grids

The `grid` element tells Mulitgraph to draw a grid behind the plot. More documentation for this will be written soon.

4.6.6. Constant Lines

Sometimes it is helpful to add a horizontal line to a graph that appear at a certain place along the axes but that doesn't correspond to a variable in your data. For example, on a temperature plot you might want to draw a horizontal line that indicates a freezing point. You can do this by creating a separate `plot` element for the line, and using the `constant` element inside the plot's `verticalaxis` element, instead of the `variable` element. The `constant` element takes one attribute, `value`, which is the data value at which the horizontal line should appear. You can specify styling details for the horizontal line, such as what plot style and colors to use, in the normal way for all plots as described in Section 4.4, “Plots”, by using the `renderer` subelement.

Chapter 5. Working with the Multigraph Source Code

Multigraph is an open source project. Anyone is welcome to download the source code and modify it. You can download the source code from http://www.multigraph.org/download.

Multigraph is written in ActionScript, which is the language of the Adobe Flash player. The Multigraph source code is organized as a project for the Flex Builder Eclipse (http://www.eclipse.org) plugin from Adobe, which is available from http://www.adobe.com/products/flex. The Multigraph source code zip file includes the Eclipse project files, so importing the project into your copy of Eclipse should be straightforward. The Flex Builder plugin is unfortunately not free, but you can download a free 60-day trial version from the above web page. Also, it IS free for use in academic institutions (https://freeriatools.adobe.com/flex), and there is an alpha version for Linux that is free to anyone (http://labs.adobe.com/technologies/flex/flexbuilder_linux).

In addition to Flex Builder, it is also possible to compile Multigraph with Adobe's free SDK compiler, which is available from http://www.adobe.com/products/flex/flexdownloads/index.html. If you are going to do much work with the Multigraph source code, though, you will probably want to get Flex Builder.

If you have any questions about working with the Multigraph source code, send an email to the multigraph-users mailing list, which is described in Chapter 8, *Obtaining Support / Mailing List*.

5.1. Adding New Renderers

"Renderers" are objects in the multigraph source code that are responsible for drawing data plots. As described in Section 4.4, “Plots”, the `renderer` element for each plot in a mugl file indicates which renderer multigraph should use for drawing that plot. New renderers that implement new plot styles can be added to multigraph by creating a new ActionScript class. This section describes how to do that.

Multigraph's renderer objects are located in the "renderer" package in the "src" folder. The "Renderer" class is an abstract base class from which all renderers descend. To create a renderer to implement a new plot style, you need to create a new subclass of the "Renderer" class. The easiest way to do that is to copy one of the existing renderer subclasses and modify it. Or at least use and existing one as a guide.

The general idea behind the "Renderer" class design is that each subclass should:

1. Define a constructor that takes two Axis arguments, calls the superclass constructor with these arguments, and does whatever else is needed in the subclass constructor.
2. Define the 3 public static string variables "keyword", "description", and "options"; these are used to autogenerate the documentation for the renderer; they are described below.
3. Define private variables corresponding to the options for this renderer, and corresponding getter/setter methods for those variables.
4. Override the 3 methods "begin()", "dataPoint()", and "end()". These are the methods that do the actual plotting; they are described below.
5. Add the name of the new subclass to the list of renderer subclasses in the "rendererList()" function definition in the file "Renderer.as".

The variables "keyword", "description", and "options" are used to autogenerate documentation for the renderer. The "keyword" variable is used by Multigraph's XML parser as the keyword for the renderer --- it is the word that should appear as the value of the `type` attribute of the `renderer` element when you want to use this renderer in a plot. The "description" string should be a short phrase that describes what this renderer does, and the "options" string should be an HTML-formatted list giving each option for the renderer, and its meaning.

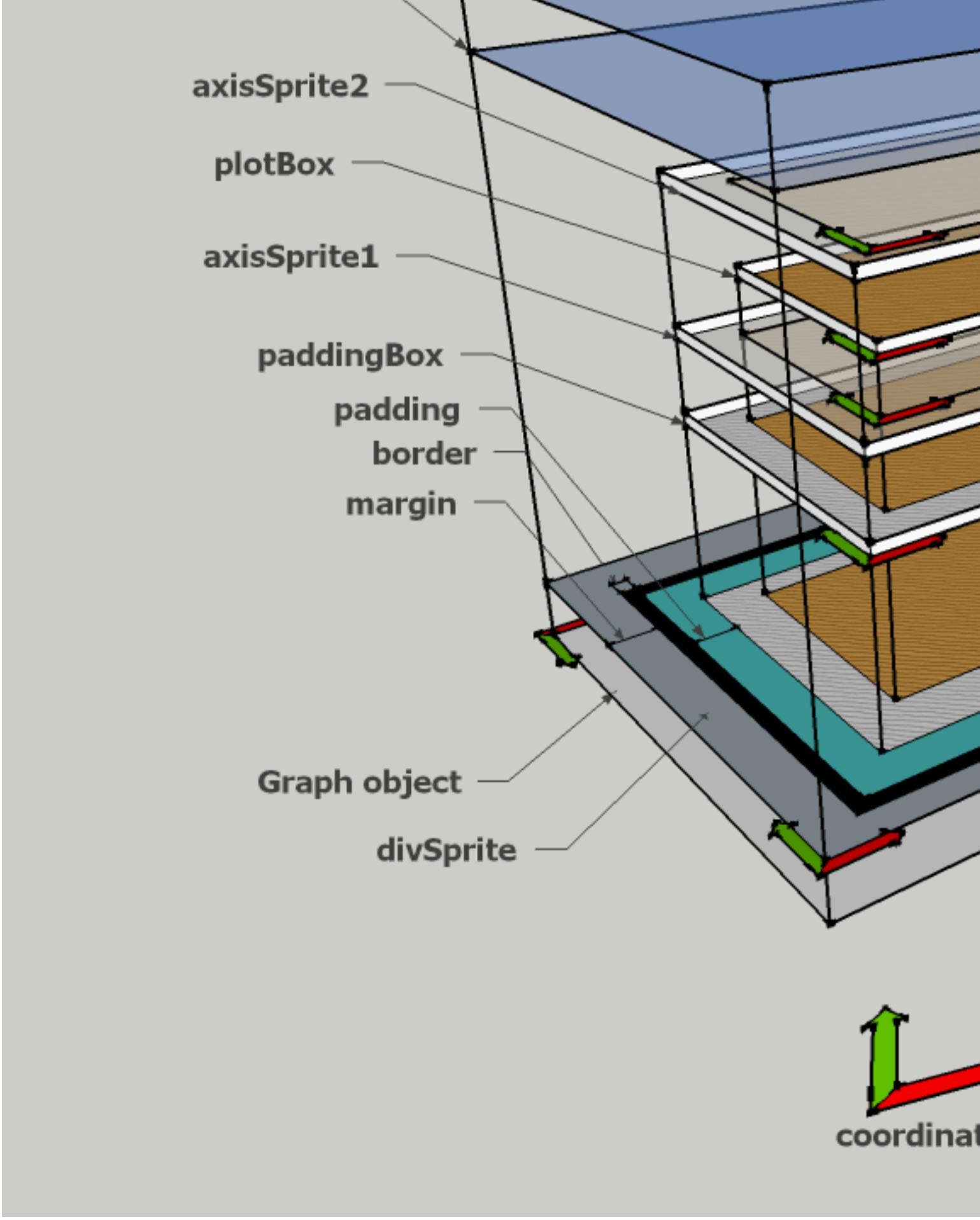
The methods "begin()", "dataPoint()", and "end()" contain the code that does the actual plotting. When Multigraph is preparing to draw a plot using a renderer, it calls the renderer's "begin()" method once. It then calls the "dataPoint()" method once for

each data point to be plotted. After all the data points to be plotted have been given by calls to "dataPoint()", Multigraph calls the "end()" method once. The "begin()" method is where the renderer can do any prep work that is needed before drawing a plot, and the "end()" method is where any needed cleanup work happens. In a simple renderer, the "dataPoint()" method might just plot the specified point, and the "begin()" and "end()" methods might not do anything (in which case they can be omitted from the renderer subclass). In general, however, the actual plotting can happen in any combintation of the "begin()", "dataPoint()", and "end()" methods. For example, in a more sophisticated renderer, the "begin()" method might initialize an array of points to be empty, the "dataPoint()" method could append each given point to that array, and the "end()" method could then loop through the points in the array, making some kind of plot that encorporates all the points at once.

5.2. The Flex Scene Graph

Multigraph uses a heirarchy of Flex objects to construct its graph. If you are working with the Multigraph source code it can be helpful to understand this heirarchy. (This information is only needed if you are working with the Multigraph source code.)

Each graph in Multigraph consists of an instance of the Graph class, which extends the Flex class `mx.core.UIComponent` and is implemented in the source file `Graph.as`. Each Graph object defines the nested heirarchy of UIComponent objects depicted in Figure 5.1, “The Scene Graph”. This documentation will eventually contain more detail about this SceneGraph.


axisSprite2
plotBox
axisSprite1
paddingBox
padding
border
margin
Graph object
divSprite
coordinat

Chapter 6. Writing Web Service Data Sources

As described in Section 4.5, “Data”, one of the ways that multigraph can obtain data to be plotted is from a "web service". A web service is an http-based API that can be used to request data from a server on the fly. Each time multigraph needs to display a plot, it will construct a URL to fetch the data needed for the plot, generate an http request using that URL, receive the response from the server, and extract the needed data from the response. It caches data received from the server so that the same data is never fetched more than once.

To write a new web service data source for multigraph, you need to create a program (cgi script, php script, web application, etc) on your server that accepts URLs using a specific syntax, and generates responses in a particular format that multigraph understands. The syntax for the URLs is the following:

```
http://www.yourserver.com/path/to/script/Xmin,Xmax/Bmin,Bmax
```

where Xmin and Xmax are values of a horizontal axis variable corresponding to the region of data being requested, and Bmin and Bmax are "buffer" values, which are explained below.

The response to the above request should be an xml-formatted response like the following:

```
<mugl>
  <data>
    <values>
       ...
    </values>
  </data>
</mugl>
```

In this response, the "..." should be replaced with a list of data values, with one set of values per line, and values on each line separated from each other by commas.

Note: as of Multigraph version 2.2-32, the XML formatting of the response is optional. The response may consist simply of the comma separated list of values, with no XML tags.

The range of data that should be returned in the response is determined by the Xmin,Xmax and Bmin,Bmax values given in the request URL. Xmin and Xmax determine an interval along the first variable dimension, and Bmin and Bmax determine an amount of "buffering" around that interval. This is easiest to explain first using the case where Bmin and Bmax are both 0. In that case, the returned list of values should start with the first available value greater than or equal to Xmin, and end with the last available value less than or equal to Xmax. If Bmin is greater than 0, then the returned list should include the next Bmin additional values less than Xmin, and if Bmax is greater than 0, the list should include Bmax additional values greater than Xmax. (Bmin and Bmax will always be nonnegative integers.)

Multigraph needs this buffering capability in its web service request syntax so that it can fetch a small amount of data on either side of a specified range. It needs this additional data in order to be able to plot the left and right edges of a data plot using renderers that involve drawing something between consecutive data points, such as the line renderer, which draws line segments connecting data points. For example, in order to be able to draw the visible piece of the line segment between the last visible data point at the right edge of a plot and the next (non-visible) point beyond the right edge, Multigraph needs to know the value of that next data point. Multigraph will usually choose buffer amounts Bmin and Bmax to both be 1, because usually one extra data point is all that is needed. In fact, at the time of this writing, Multigraph always uses Bmin and Bmax values of 1. It is possible that in the future, however, there could be situations where greater buffer values are needed, so it is best to design any Multigraph web data services that you write to be able to handle arbitrary Bmin and Bmax values.

A couple examples will hopefully clarify this. In the first example, imagine a data set that consists of hourly temperature values for the month of January 2009 --- one temperature value corresponding to the top of each hour --- and imagine that Multigraph is preparing to draw a plot using the line renderer for the 6 hour period starting at 5:30am and ending at 11:30am on January 12. There are 6 data points visible in that time range, corresponding to times 6:00, 7:00, 8:00, 9:00, 10:00, and 11:00. Multigraph will also need one more data point on either side of this range, though, namely the data for 5:00 and 12:00, in order to be able to draw the partially visible line segments between the 11:00 value and the right edge of the graph, and between the 6:00 value and the left edge of the graph. Since the actual interval being displayed ranges from 5:30 to 11:30, the Xmin value in the request that Multigraph generates will correspond to the time 5:30, and the Xmax value will correspond to 11:30. The Bmax and Bmin values will both be 1. The generated request URL therefore looks like the following:

```
http://www.yourserver.com/path/to/script/200901120530,200901121130/1,1
```

The response to the above request might look like the following

```
<mugl>
  <data>
    <values>
200901120500,22.1
200901120600,23.4
200901120700,24.2
200901120800,23.1
200901120900,25.8
200901121000,27.9
200901121100,28.2
200901121200,29.3
    </values>
  </data>
</mugl>
```

In this example, Multigraph is drawing a plot that shows 6 data values, but in order to be able to completely draw that plot, it actually needs 2 additional data values, so the request includes a buffer of 1 on either side of the requested interval, and the response includes 8 values rather than 6. If you are wondering why the buffering can't be accomplished by simply having Multigraph request an 8 hour interval in the first place, and eliminate the need for the extra Bmin and Bmax request parameters altogether, the answer is that Multigraph does not know that the data freqency is hourly. When Multigraph plots data from a web service, it relies completely on the web service to provide the data, and makes no assumptions about how much data is present in a particular interval. It is up to the web service to know that the data occurrs hourly, or every 5 minutes, or daily, or whatever.

Chapter 7. Using the Multigraph Flex Component

If you are writing your own Flex application, you can use Multigraph as a component in your application. To do so, first download the Multigraph Flex Component library file, named Multigraph-*.swc (where * is the version number), from http://www.multigraph.org/download. If you are using Flex Builder, add this SWC file to your project library path; if you are using the Flex SDK, use the relevant options to add the file to your mxmlc command.

In your application, you can create a Multigraph graph using the `Multigraph` tag in your mxml file. This tag takes an attribute called `muglfile` which is the name or URL of the mugl file for your graph. For example:

```
<mx:Application
  xmlns:mx="http://www.adobe.com/2006/mxml"
  xmlns:multigraph="multigraph.*"
  horizontalAlign="left"
  paddingLeft="0"
  paddingTop="0"
  paddingBottom="0"
  paddingRight="0">
  <multigraph:Multigraph muglfile="graph.xml"/>
</mx:Application>
```

Unless your mugl file is specified as a URL to a different server, be sure to deploy your mulgl file to your server along with your application's SWF file.

Note that when using the Multigraph component in a Flex application you do not need to create the HTML file described in Chapter 2, *Quick Start* or Chapter 3, *Creating Multigraph Web Pages*, and you do not need to deploy Multigraph to your web server separately from your application. The Multigraph SWC file, which is compiled into your application, includes everything that Multigraph needs to work in your application, with the exception of your mugl file(s).

The `Multigraph` component extends the `mx.containers.VBox` object, so it inherits all the tag attributes of that object, which may be used to further control its appearance and behavior.

Chapter 8. Obtaining Support / Mailing List

The multigraph-users mailing list is the place to turn for questions about multigraph. It is also the best way to contact the authors. The home page for the list is http://groups.google.com/group/multigraph-users; you can access the list archives, and find instructions on how to join the list, on that page.

Chapter 9. Credits

Multigraph has been written by Mark Phillips and Devin Eldreth at the University of North Carolina at Asheville and NOAA's National Climatic Data Center.

Chapter 10. License

Multigraph is distributed under the terms of the following copyright and license.

understand the U.S. export regulations. Licensees in other countries may face similar restrictions. In all cases, it is licensee's responsibility to comply with any export regulations applicable in licensee's jurisdiction.