12

Plot Package

Authors: *Edward A. Lee*
Christopher Hylands
Contributors: *Lukito Muliadi*
William Wu
Jun Wu

12.1 Overview

The plot package provides classes, applets, and applications for two-dimensional graphical display of data. It is available in a stand-alone distribution, or as part of the Ptolemy II system.

There are several ways to use the classes in the plot package:

- You can use one of several domain-polymorphic actors in a Ptolemy II model to plot data that is provided as an input to the actor.
- You can invoke an executable, `ptplot`, which is a shell script, to plot data in a local file or on the network (via a URL).
- You can invoke an executable, `histogram`, which is a shell script, to plot histograms of data in a local file or on the network (via a URL)
- You can invoke an executable, `pxgraph`, which is a shell script, to plot data that is stored in an ascii or binary format compatible with the older program `pxgraph`, which is an extension of David Harrison's `xgraph`.
- You can invoke a Java application, such as PlotMLApplication, by using the `java` program that is included in your Java distribution.
- You can use an existing applet class, such as PlotMLApplet, in an HTML file. The applet parameter `dataurl` specifies the source of plot data. You do not even have to have Ptplot installed on your server, since you can always reference the Berkeley installation.
- You can create new classes derived from applet, frame, or application classes to customize your

plots. This allows you to completely control the placement of plots on the screen, and to write Java code that defines the data to be plotted.

The plot data can be specified in any of three data formats:

- *PlotML* is an XML extension for plot data. Its syntax is similar to that of HTML. XML (extensible markup language) is an internet language that is growing rapidly in popularity.
- An older, simpler textual syntax for plot data is also provided, although in the long term, that syntax is unlikely to be maintained (it will not necessarily be expanded to support new features). For simple data plots, however, it is adequate. Using it for applets has the advantage of making it possible to reference a slightly smaller jar file containing the code, which makes for more responsive applets. Also, the data files are somewhat smaller.
- A binary file format used by `pxgraph`, is supported by classes in the compat package. Formatting information in `pxgraph` (and in the compat package) is provided by command-line arguments, rather than being included with the binary plot data, exactly as in the older program. Applets specify these command-line arguments as an applet parameter (pxgraphargs).

12.2 Using Plots

If $PTII represents the home directory of your Ptplot installation (or your Ptolemy II installation), then, $PTII/bin is a directory that contains a number of executables. Three of these invoke plot applications, `ptplot`, `histogram`, and `pxgraph`. We recommend putting this directory into your path so that these executables can be found automatically from the command line. Invoking the command

```
ptplot
```

with no arguments should open a window that looks like that in figure 12.1. You can also specify a file to plot as a command-line argument. To find out about command-line options, type

```
ptplot -help
```

The ptplot command is a shell script that invokes the following equivalent command:

```
java -classpath $PTII ptolemy.plot.plotml.EditablePlotMLApplication
```

Since it is a shell script, it will work on Unix machines and Windows machines that have Cygwin[1] installed. In the same directory are three Windows versions that do not require Cygwin, `ptplot.bat`, `histogram.bat`, and `pxgraph.bat`, which you can invoke by typing into the DOS command prompt, for example,

```
ptplot.bat
```

1. The 1.1.x version of the Cygwin Toolkit is a freely available package available from http://sources.redhat.com/cygwin/

These scripts make three assumptions.

- First, `java` is in your path. Type "`java -version`" to verify that the `java` program is in your path and is working properly
- Second, the environment variable PTII is set to point to the home directory of the plot (or Ptolemy II) installation. Type "`echo %PTII%`" in a Windows DOS shell and "`echo $PTII`" in Unix or Windows Cygwin bash shell to check this.
- The directory $PTII/bin is in your path. Under Windows without Cygwin, type "`echo %PATH%`". Type "`type ptplot`" in Windows with Cygwin and "`which ptplot`" in Unix to check this.

In Windows, environment variables and your path are set in the System control panel. You can now explore a number of features of ptplot.

12.2.1 Zooming and filling

To zoom in, drag the left mouse button down and to the right to draw a box around an area that you want to see in detail, as shown in figure 12.2. To zoom out, drag the left mouse button up and to the right. To just fill the drawing area with the available data, type Control-F, or invoke the fill command from the Special menu. In applets, since there is no menu, the fill command is (optionally) made available as a button at the upper right of the plot.

12.2.2 Printing and exporting

The File menu includes a Print and Export command. The Print command works as you expect. The export command produces an encapsulated PostScript file (EPS) suitable for inclusion in word processors. The image in figure 12.3 is such an EPS file imported into FrameMaker.

At this time, the EPS file does not include preview data. This can make it somewhat awkward to work with in a word processor, since it will not be displayed by the word processor while editing (it

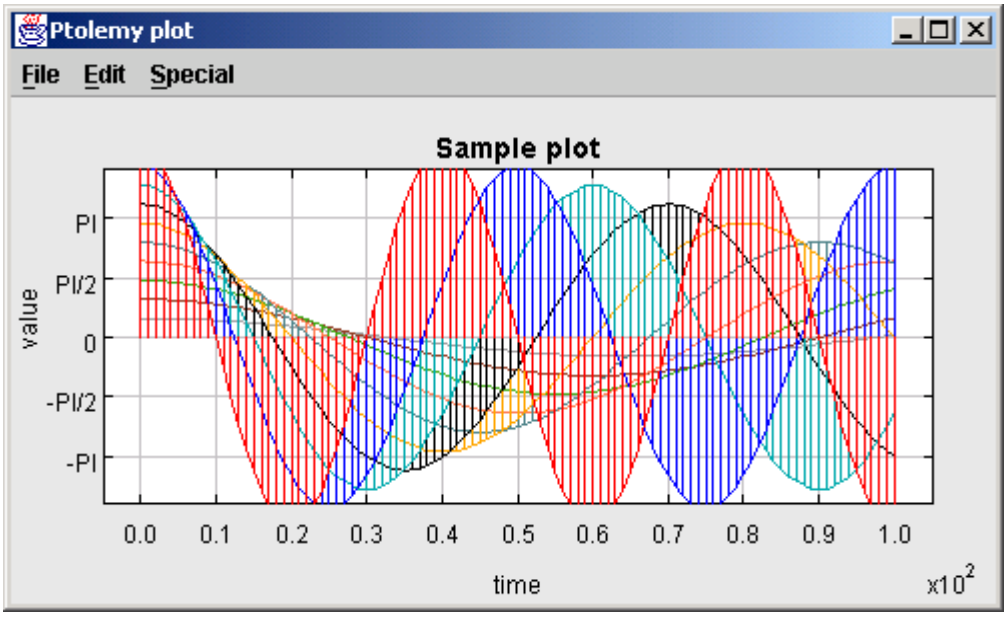

FIGURE 12.1. Result of invoking *ptplot* on the command line with no arguments.

will, however, print correctly). It is easy to add the preview data using the freely available program Ghostview[1]. Just open the file using Ghostview and, under the edit menu, select “Add EPS Preview.”

Export facilities are also available from a small set of key bindings, which permits them to be invoked from applets (which have no menu bar) and from the standalone scripts:

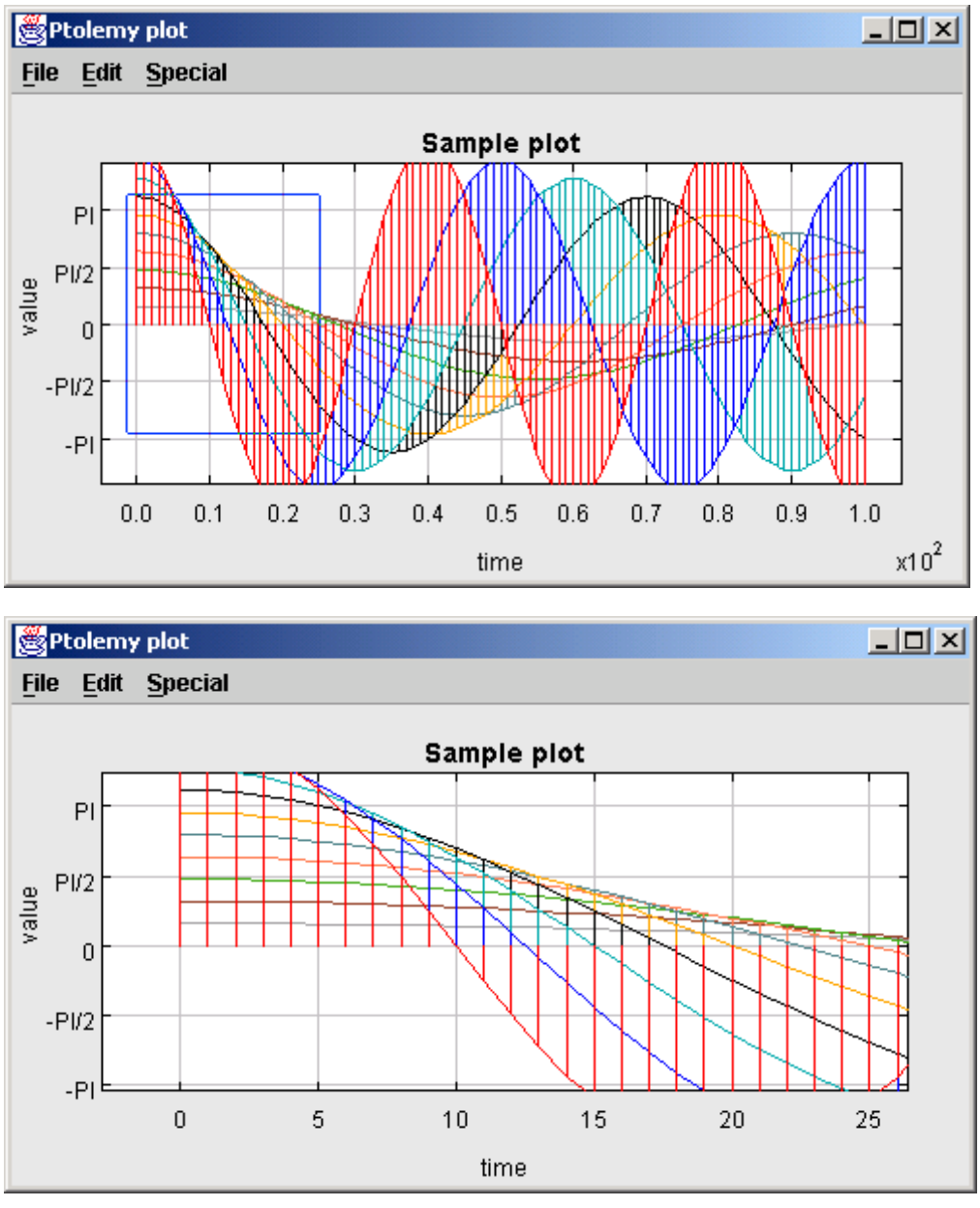

FIGURE 12.2. To zoom in, drag the left mouse button down and to the right to draw a box around the region you wish to see in more detail.

1. Ghostview is available http://www.cs.wisc.edu/~ghost

- Control-c: Export the plot to the clipboard.
- D: Dump the plot to standard output.
- E: Export the plot to standard output in EPS format.
- F: Fill the plot.
- H or ?: Display a simple help message.

The encapsulated PostScript (EPS) that is produced is tuned for black-and-white printers. In the future, more formats may supported. Also at this time (JDK 1.3.0 under Windows 2000), Java's interface the clipboard may not work, so Control-C might not accomplish anything. Note further that with applets, you may find it best to click near the title rather than clicking inside the graph itself and then type the command.

Exporting to the clipboard and to standard output, in theory, is allowed for applets, unlike writing to a file. Thus, these key bindings provide a simple mechanism to obtain a high-resolution image of the plot from an applet, suitable for incorporation in a document. However, in some browsers, exporting to standard out triggers a security violation. You can use Sun's appletviewer instead.

12.2.3 Editing the data

You can modify the data that is plotted by first selecting a data set to modify using the Edit menu, then dragging the right mouse button. Figure 12.4 shows the result of modifying one of the datasets (the one in red on a color display). The modification is carried out by freehand drawing, although considerable precision is possible by zooming in. Use the Save or SaveAs command in the File menu to save the modified plot (in PlotML format).

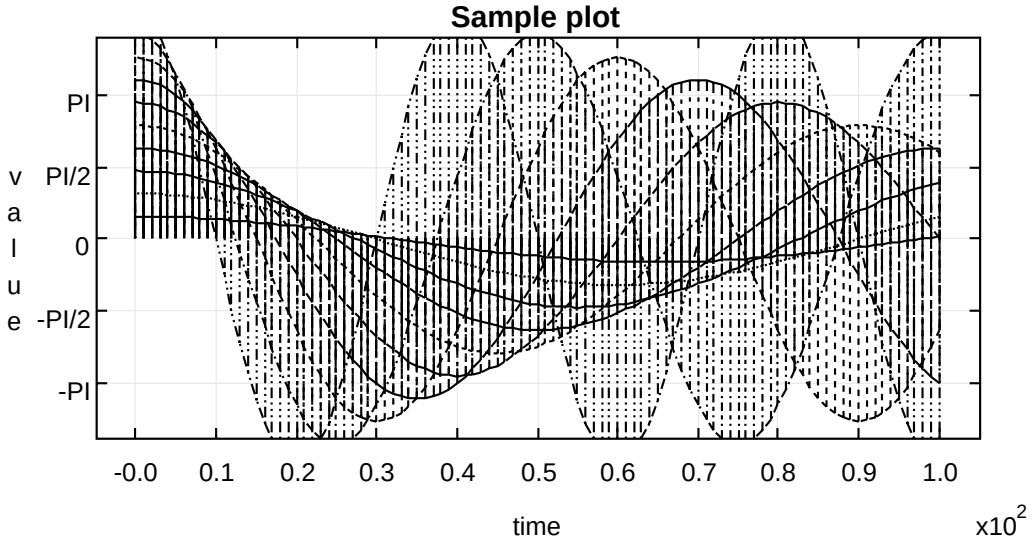

FIGURE 12.3. Encapsulated postscript generated by the Export command in the File menu of *ptplot* can be imported into word processors. This figure was imported into FrameMaker.

12.2.4 Modifying the format

You can control how data is displayed by invoking the Format command in the Edit menu. This brings up a dialog like that at bottom in figure 12.5. At the left is the dialog and the plot before changes are made, and at the right is after changes are made. In particular, the grid has been removed, the stems have been removed, the lines connecting the data points have been removed, the data points have been rendered with points, and the color has been removed. Use the Save or SaveAs command in the File menu to save the modified plot (in PlotML format). More sophisticated control over the plot can be had by editing the PlotML file (which is a text file). The PlotML syntax is described below.

The entries in the format dialog are all straightforward to use except the "X Ticks" and "Y Ticks" entries. These are used to specify how the axes are labeled. The tick marks for the axes are usually computed automatically from the ranges of the data. Every attempt is made to choose reasonable positions for the tick marks regardless of the data ranges (powers of ten multiplied by 1, 2, or 5 are used). To change what tick marks are included and how they are labeled, enter into the "X Ticks" or "Y Ticks" entry boxes a string of the following form:

```
label position, label position, ...
```

A *label* is a string that must be surrounded by quotation marks if it contains any spaces. A position is a number giving the location of the tick mark along the axis. For example, a horizontal axis for a frequency domain plot might have tick marks as follows:

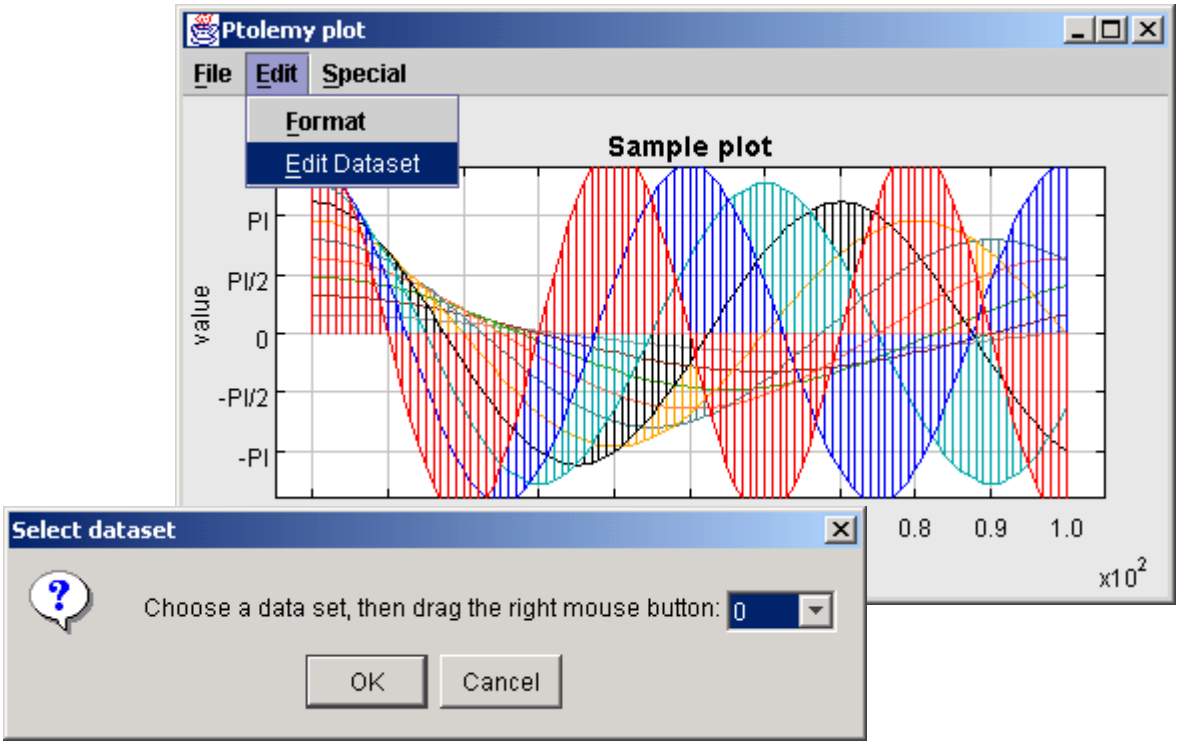

FIGURE 12.4. You can modify the data being plotted by selecting a data set and then dragging the right mouse button. Use the Edit menu to select a data set. Use the Save command in the File menu to save the modified plot (in PlotML format).

```
XTicks: -PI -3.14159, -PI/2 -1.570795, 0 0, PI/2 1.570795, PI 3.14159
```

Tick marks could also denote years, months, days of the week, etc.

12.3 Class Structure

The plot package has two subpackages, plotml and compat. The core package, plot, contains toolkit classes, which are used in Java programs as building blocks. The two subpackages contain classes that are usable by an end-user (vs. a programmer).

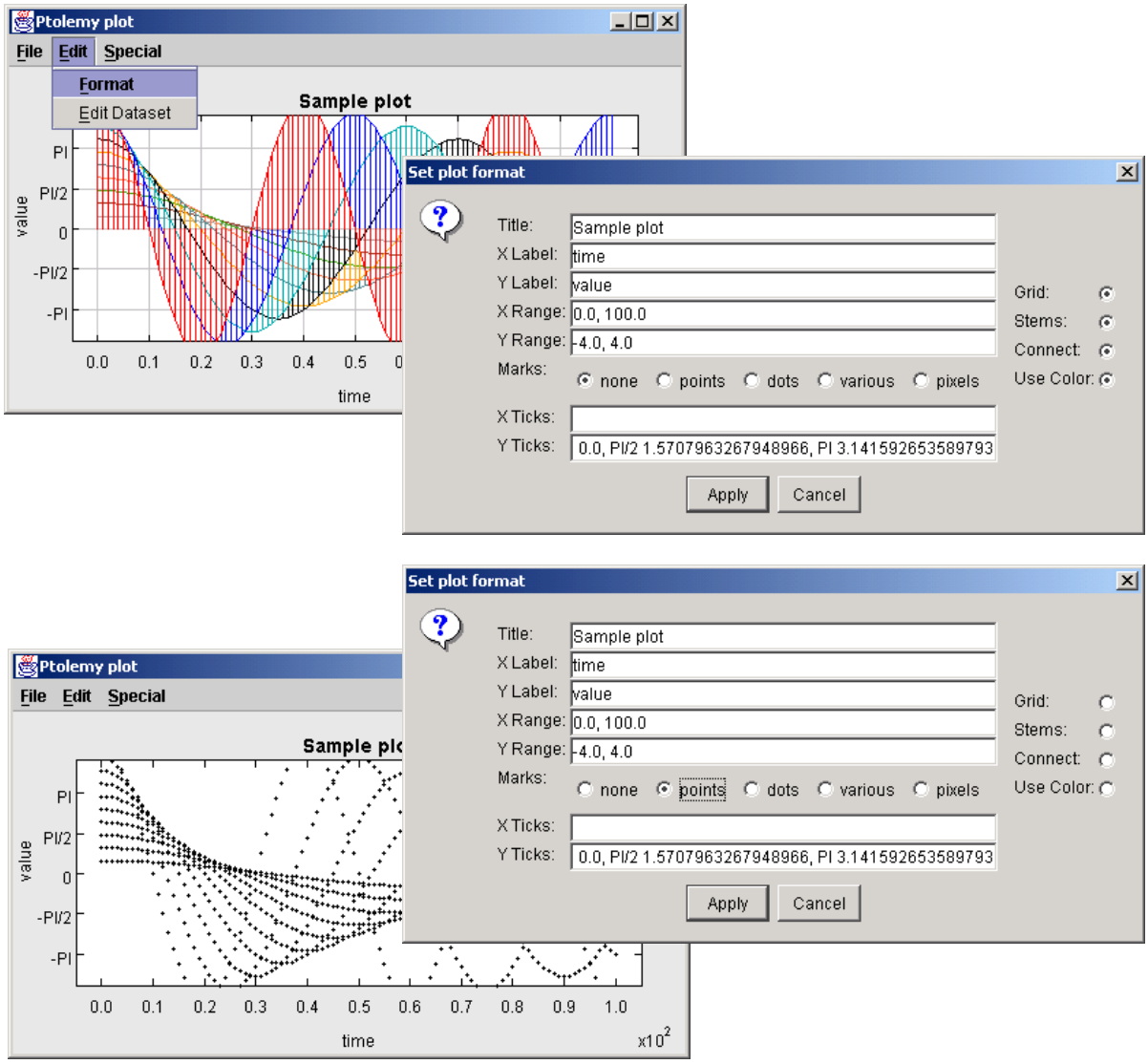

FIGURE 12.5. You can control how data is displayed using the Format command in the Edit menu, which brings up the dialog shown at the right. On the top is before changes are made, and on the bottom is after.

12.3.1 Toolkit classes

The class diagram for the core of the plot package is shown in figure 12.6. These classes provide a toolkit for constructing plotting applications and applets. The base class is PlotBox, which renders the axes and the title. It extends Panel, a basic container class in Java. Consequently, plots can be incorporated into virtually any Java-based user interface.

The Plot class extends PlotBox with data sets, which are collections of instances of PlotPoint. The EditablePlot class extends this further by adding the ability to modify data sets.

Live (animated) data plots are supported by the PlotLive class. This class is abstract; a derived class must be created to generate the data to plot (or collect it from some other application).

The Histogram class extends PlotBox rather than Plot because many of the facilities of Plot are irrelevant. This class computes and displays a histogram from a data file. The same data file can be read by this class and the other plot classes, so you can plot both the histogram and the raw data that is used to generate it from the same file.

12.3.2 Applets and applications

A number of classes are provided to use the plot toolkit classes in common ways, but you should keep in mind that these classes are by no means comprehensive. Many interesting uses of the plot package involve writing Java code to create customized user interfaces that include one or more plots. The most commonly used built-in classes are those in the *plotml* package, which can read PlotML files, as well as the older textual syntax.

Ptplot 5.2, which shipped with Ptolemy II 2.0 requires Swing. The easiest way to get Swing is to install the Java 1.3 (or later) Plug-in, which is part of the JRE and JDK 1.3 installation. Unfortunately, using the Java Plug-in makes the applet HTML more complex. There are two choices:

1. Use fairly complex JavaScript to determine which browser is running and then to properly select one of three different ways to invoke the Java Plug-in. This method works on the most different types of platforms and browsers. The JavaScript is so complex, that rather than reproduce it here, please see one of the demonstration html files.

2. Use the much simpler <applet> ...</applet> tag to invoke the Java Plug-in. This method works on many platforms and browsers, but requires a more recent version of the Java Plug-in, and will not work under Netscape Communicator 4.7x.

   For details about the above two choices, see http://java.sun.com/products/plugin/versions.html.

   We document the much simpler <applet> . . . </applet> tag format below

   The following segment of HTML is an example:

```
<APPLET
 code = "ptolemy.plot.plotml.PlotMLApplet"
 codebase = "../../.."
 archive = "ptolemy/plot/plotmlapplet.jar"
 width = "600"
 height = "400"
 >
<PARAM NAME = "background" VALUE = "#faf0e6" >
<PARAM NAME = "dataurl" VALUE = "plotmlSample.txt" >
  No Java Plug-in support for applet, see
 <a href="http://java.sun.com/products/plugin/"><code>http://java.sun.com/products/plugin/</code></a>
</APPLET>
```

To use this yourself you will probably need to change the *codebase* and *dataurl* entries. The first points

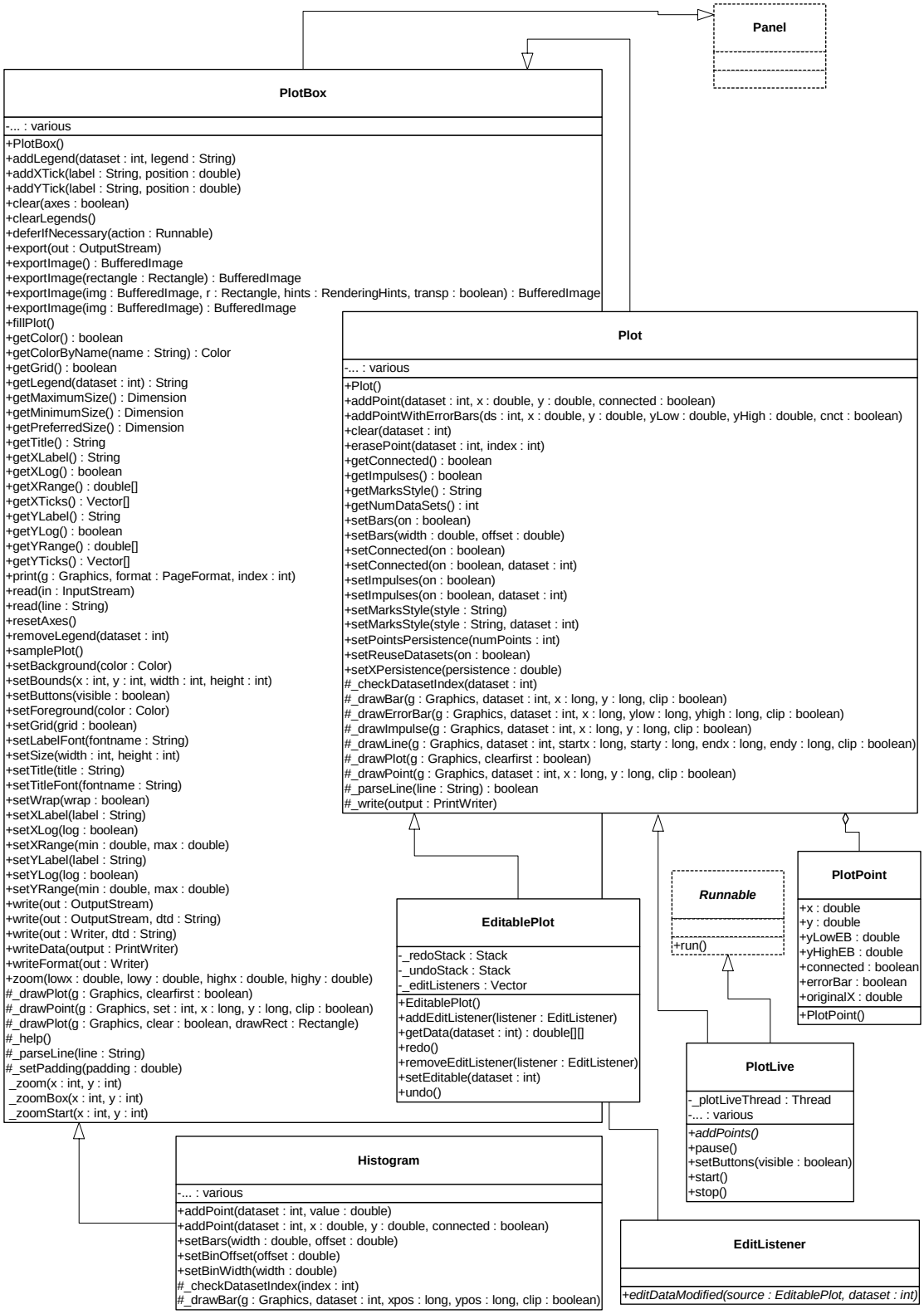

FIGURE 12.6. The core classes of the plot package.

to the root directory of the plot installation (usually, the value of the PTII environment variable). The second points to a file containing data to be plotted, plus optional formatting information. The file format for the data is described in the next section. The applet is created by instantiating the PlotMLApplet class.

The *archive* entry contains the name of the jar file that contains all the classes necessary to run a PlotML applet. The advantage of specifying a jar file is that remote users are likely to experience a faster download because all the classes come over at once, rather than the browser asking for each class from the server. A downside of using jar files in applets is that if you are modifying the source of Ptplot itself, then you must also update the jar file, or your changes will not appear. A common workaround is to remove the archive entry during testing.

You can also easily create your own applet classes that include one or more plots. As shown in figure 12.6, the PlotBox class is derived from JPanel, a basic class of the Java Foundation Classes (JFC) toolkit, also known as swing. It is easy to place a panel in an applet, positioned however you like, and to combine multiple panels into an applet. PlotApplet is a simple class that adds an instance of Plot.

Creating an application that includes one or more plots is also easy. The PlotApplication class, shown in figure 12.7, creates a single top-level window (a JFrame), and places within it an instance of Plot. This class is derived from the PlotFrame class, which provides a menu that contains a set of commands, including opening files, saving the plotted data to a file, printing, etc.

The difference between PlotFrame and PlotApplication is that PlotApplication includes a main() method, and is designed to be invoked from the command line. You can invoke it using commands like the following:

```
java -classpath $PTII ptolemy.plot.PlotApplication args
```

However, the classes shown in figure 12.7, which are in the plot package, are not usually the ones that an end user will use. Instead, use the ones in figure 12.8. These extend the base classes to support the PlotML language, described below. The only motivation for using the base classes in figure 12.7 is to have a slightly smaller jar file to load for applets.

The classes that end users are likely to use, shown in figure 12.8, include:

- PlotMLApplet: An applet that can read PlotML files off the web and render them.
- EditablePlotMLApplet: A version that allows editing of any data set in the plot.
- HistogramMLApplet: A version that uses the Histogram class to compute and plot histograms.
- PlotMLFrame: A top-level window containing a plot defined by a PlotML file.
- PlotMLApplication: An application that can be invoked from the command line and reads PlotML files.
- EditablePlotMLApplication: An extension that allows editing of any data set in the plot.
- HistogramMLApplication: A version that uses the Histogram class to compute and plot histograms.

EditablePlotMLApplication is the class invoked by the `ptplot` command-line script. It can open plot files, edit them, print them, and save them.

12.3.3 Writing applets

A plot can be easily embedded within an applet, although there are some subtleties. The simplest mechanism looks like this:

```
public class MyApplet extends JApplet {
  public void init() {
    super.init();
    Plot myplot = new Plot();
    getContentPane().add(myplot);
```

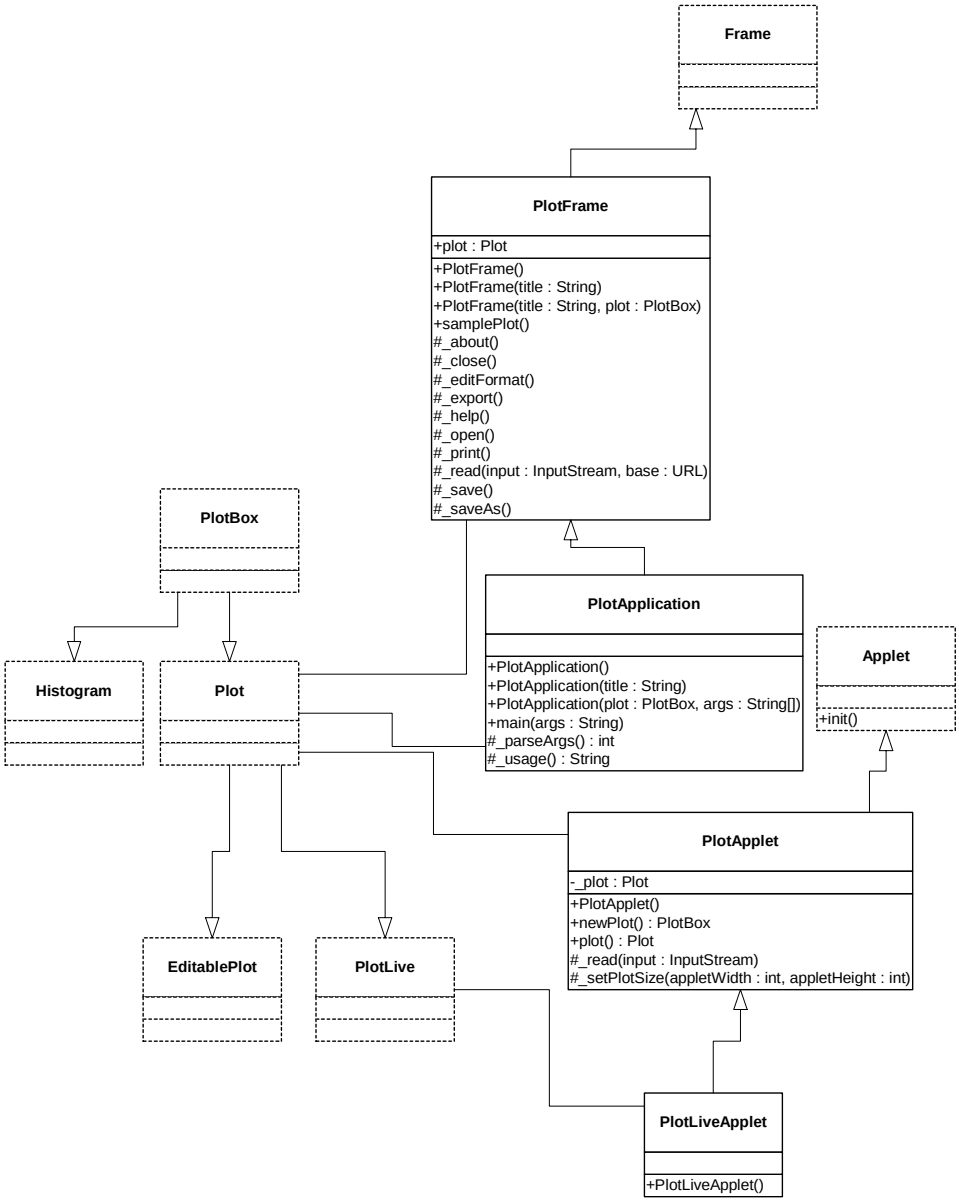

FIGURE 12.7. Core classes supporting applets and applications. Most of the time, you will use the classes in the plotml package, which extend these with the ability to read PlotML files.

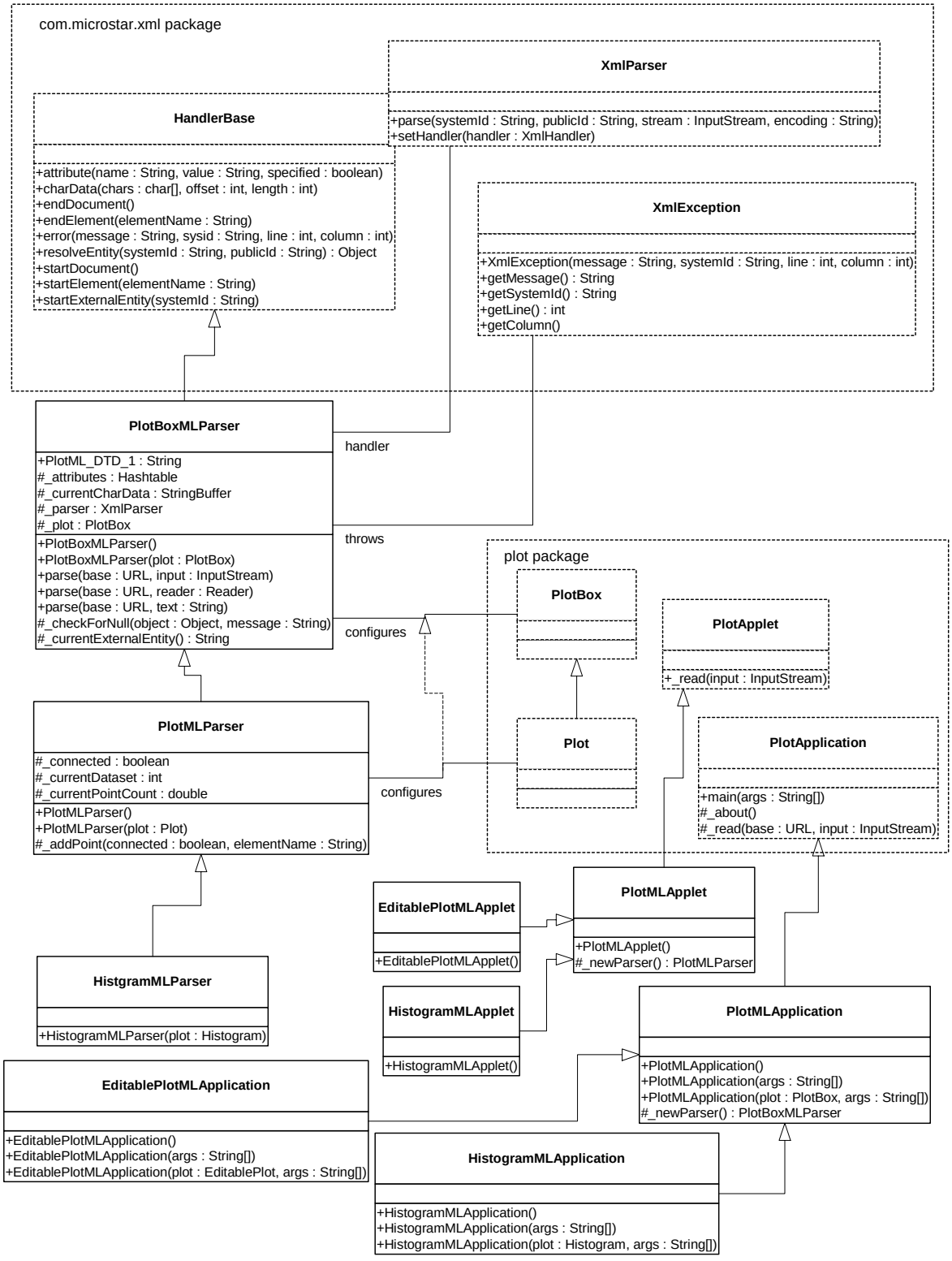

FIGURE 12.8. UML static structure diagram for the plotml package, a subpackage of plot providing classes

```
        myplot.setTitle("Title of plot");
        ...
    }
}
```

This places the plot in the center of the applet space, stretching it to fill the space available. To control the size independently of that of the applet, for some mysterious reason that only Sun can answer, it is necessary to embed the plot in a panel, as follows:

```
public class MyApplet extends JApplet {
  public void init() {
    super.init();
    Plot myplot = new Plot();
    JPanel panel = new JPanel();
    getContentPane().add(panel);
    panel.add(myplot);
    myplot.setSize(500, 300);
    myplot.setTitle("Title of plot");
    ...
  }
}
```

The setSize() method specifies the width and height in pixels. You will probably want to control the background color and/or the border, using statements like:

```
        myplot.setBackground(background color);
        myplot.setBorder(new BevelBorder(BevelBorder.RAISED));
```

Alternatively, you may want to make the plot transparent, which results in the background showing through:

```
        myplot.setOpaque(false);
```

12.4 PlotML File Format

Plots can be specified as textual data in a language called PlotML, which is an XML extension. XML, the popular *extensible markup language*, provides a standard syntax and a standard way of defining the content within that syntax. The syntax is a subset of SGML, and is similar to HTML. It is intended for use on the internet. Plot classes can save data in this format (in fact, the Save operation always saves data in this format), and the classes in the plotml subpackage, shown in figure 12.8, can read data in this format. The key classes supporting this syntax are PlotBoxMLParser, which parses a subset of PlotML supported by the PlotBox class, PlotMLParser, which parses the subset of PlotML supported by the Plot class, and HistogramMLParser, which parses the subset that supports histograms.

12.4.1 Data organization

Plot data in PlotML has two parts, one containing the plot data, including format information (how the plot looks), and the other defining the PlotML language. The latter part is called the *document type definition*, or DTD. This dual specification of content and structure is a key XML innovation.

Every PlotML file must either contain or refer to a DTD. The simplest way to do this is with the following file structure:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE model PUBLIC "-//UC Berkeley//DTD PlotML 1//EN"
    "http://ptolemy.eecs.berkeley.edu/xml/dtd/PlotML_1.dtd">
<plot>
    format commands...
    datasets...
</plot>
```

Here, "*format commands*" is a set of XML elements that specify what the plot looks like, and "*datasets*" is a set of XML elements giving the data to plot. The syntax for these elements is described below in subsequent sections. The first line above is a required part of any XML file. It asserts the version of XML that this file is based on (1.0) and states that the file includes external references (in this case, to the DTD). The second and third lines declare the document type (plot) and provide references to the DTD.

The references to the DTD above refer to a "public" DTD. The name of the DTD is

```
-//UC Berkeley//DTD PlotML 1//EN
```

which follows the standard naming convention of public DTDs. The leading dash "`-`" indicates that this is not a DTD approved by any standards body. The first field, surrounded by double slashes, in the name of the "owner" of the DTD, "`UC Berkeley`." The next field is the name of the DTD, "`DTD PlotML 1`" where the "`1`" indicates version 1 of the PlotML DTD. The final field, "`EN`" indicates that the language assumed by the DTD is English.

In addition to the name of the DTD, the `DOCTYPE` element includes a URL pointing to a copy of the DTD on the web. If a particular PlotML tool does not have access to a local copy of the DTD, then it finds it at this web site. PtPlot recognizes the public DTD, and uses its own local version of the DTD, so it does not need to visit this website in order to open a PlotML file.

An alternative way to specify the DTD is:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE plot SYSTEM "DTD location">
<plot>
    format commands...
    datasets...
</plot>
```

Here, the DTD location is a relative or absolute URL.

A third alternative is to create a standalone PlotML file that includes the DTD. The result is rather verbose, but has the general structure shown below:

```
<?xml version="1.0" standalone="yes"?>
<!DOCTYPE plot [
    DTD information
]>
<plot>
    format commands
    datasets
</plot>
```

These latter two methods are useful if you extend the DTD.

The DTD for PlotML is shown in figure 12.9. This defines the PlotML language. However, the DTD is not particularly easy to read, so we define the language below in a more tutorial fashion.

12.4.2 Configuring the axes

The elements described in this subsection are understood by the base class PlotBoxMLParser.

```
<title>Your Text Here</title>
```

The title is bracketed by the start element `<title>` and end element `</title>`. In XML, end elements are always the same as the start element, except for the slash. The DTD for this is simple:

```
<!ELEMENT title (#PCDATA)>
```

This declares that the body consists of *PCDATA, parsed character data.*

Labels for the X and Y axes are similar,

```
<xLabel>Your Text Here</xLabel>
<yLabel>Your Text Here</yLabel>
```

Unlike HTML, in XML, case is important. So the element is `xLabel` not `XLabel`.

The ranges of the X and Y axes can be optionally given by:

```
<xRange min="min" max="max"/>
<yRange min="min" max="max"/>
```

The arguments *min* and *max* are numbers, possibly including a sign and a decimal point. If they are not specified, then the ranges are computed automatically from the data and padded slightly so that datapoints are not plotted on the axes. The DTD for these looks like:

```
<!ELEMENT xRange EMPTY>
    <!ATTLIST xRange min CDATA #REQUIRED
                     max CDATA #REQUIRED>
```

The EMPTY means that the element does not have a separate start and end part, but rather has a final slash before the closing character "`/>`". The two ATTLIST elements declare that min and max

```
<!ELEMENT plot (barGraph | bin | dataset | default | noColor | noGrid | size | title | wrap | xLabel |
   xLog | xRange | xTicks | yLabel | yLog | yRange | yTicks)*>
  <!ELEMENT barGraph EMPTY>
    <!ATTLIST barGraph width CDATA #IMPLIED
                       offset CDATA #IMPLIED>
  <!ELEMENT bin EMPTY>
    <!ATTLIST bin width CDATA #IMPLIED
                  offset CDATA #IMPLIED>
  <!ELEMENT dataset (m | move | p | point)*>
    <!ATTLIST dataset connected (yes | no) #IMPLIED
                      marks (none | dots | points | various | pixels) #IMPLIED
                      name CDATA #IMPLIED
                      stems (yes | no) #IMPLIED>
  <!ELEMENT default EMPTY>
    <!ATTLIST default connected (yes | no) "yes"
                      marks (none | dots | points | various | pixels) "none"
                      stems (yes | no) "no">
  <!ELEMENT noColor EMPTY>
  <!ELEMENT noGrid EMPTY>
  <!ELEMENT reuseDatasets EMPTY>
  <!ELEMENT size EMPTY>
    <!ATTLIST size height CDATA #REQUIRED
                   width CDATA #REQUIRED>
  <!ELEMENT title (#PCDATA)>
  <!ELEMENT wrap EMPTY>
  <!ELEMENT xLabel (#PCDATA)>
  <!ELEMENT xLog EMPTY>
  <!ELEMENT xRange EMPTY>
    <!ATTLIST xRange min CDATA #REQUIRED
                     max CDATA #REQUIRED>
  <!ELEMENT xTicks (tick)+>
  <!ELEMENT yLabel (#PCDATA)>
  <!ELEMENT yLog EMPTY>
  <!ELEMENT yRange EMPTY>
    <!ATTLIST yRange min CDATA #REQUIRED
                     max CDATA #REQUIRED>
  <!ELEMENT yTicks (tick)+>
    <!ELEMENT tick EMPTY>
      <!ATTLIST tick label CDATA #REQUIRED
                     position CDATA #REQUIRED>
    <!ELEMENT m EMPTY>
      <!ATTLIST m x CDATA #IMPLIED
                  y CDATA #REQUIRED
                  lowErrorBar CDATA #IMPLIED
                  highErrorBar CDATA #IMPLIED>
    <!ELEMENT move EMPTY>
      <!ATTLIST move x CDATA #IMPLIED
                     y CDATA #REQUIRED
                     lowErrorBar CDATA #IMPLIED
                     highErrorBar CDATA #IMPLIED>
    <!ELEMENT p EMPTY>
      <!ATTLIST p x CDATA #IMPLIED
                  y CDATA #REQUIRED
                  lowErrorBar CDATA #IMPLIED
                  highErrorBar CDATA #IMPLIED>
    <!ELEMENT point EMPTY>
      <!ATTLIST point x CDATA #IMPLIED
                      y CDATA #REQUIRED
                      lowErrorBar CDATA #IMPLIED
                      highErrorBar CDATA #IMPLIED>
```

FIGURE 12.9. The *document type definition* (DTD) for the PlotML language.

attributes are required, and that they consist of character data.

The tick marks for the axes are usually computed automatically from the ranges. Every attempt is made to choose reasonable positions for the tick marks regardless of the data ranges (powers of ten multiplied by 1, 2, or 5 are used). However, they can also be specified explicitly using elements like:

```
<xTicks>
  <tick label="label" position="position"/>
  <tick label="label" position="position"/>
  ...
</xTicks>
```

A *label* is a string that replaces the number labels on the axes. A position is a number giving the location of the tick mark along the axis. For example, a horizontal axis for a frequency domain plot might have tick marks as follows:

```
<xTicks>
  <tick label="-PI" position="-3.14159"/>
  <tick label="-PI/2" position="-1.570795"/>
  <tick label="0" position="0"/>
  <tick label="PI/2" position="1.570795"/>
  <tick label="PI" position="3.14159"/>
</xTicks>
```

Tick marks could also denote years, months, days of the week, etc. The relevant DTD information is:

```
<!ELEMENT xTicks (tick)+>
    <!ELEMENT tick EMPTY>
      <!ATTLIST tick label CDATA #REQUIRED
                     position CDATA #REQUIRED>
```

The notation `(tick)+` indicates that the `xTicks` element contains one or more `tick` elements.

If ticks are not specified, then the X and Y axes can use a logarithmic scale with the following elements:

```
<xLog/>
<yLog/>
```

The tick labels, which are computed automatically, represent powers of 10. The log axis facility has a number of limitations, which are documented in “Limitations” on page 12-24.

By default, tick marks are connected by a light grey background grid. This grid can be turned off with the following element:

```
<noGrid/>
```

Also, by default, the first ten data sets are shown each in a unique color. The use of color can be turned off with the element:

```
<noColor/>
```

Finally, the rather specialized element

```
<wrap/>
```

enables wrapping of the X (horizontal) axis, which means that if a point is added with X out of range, its X value will be modified modulo the range so that it lies in range. This command only has an effect if the X range has been set explicitly. It is designed specifically to support oscilloscope-like behavior, where the X value of points is increasing, but the display wraps it around to left. A point that lands on the right edge of the X range is repeated on the left edge to give a better sense of continuity. The feature works best when points do land precisely on the edge, and are plotted from left to right, increasing in X.

You can also specify the size of the plot, in pixels, as in the following example:

```
<size width="400" height="300">
```

All of the above commands can also be invoked directly by calling the corresponding public methods from Java code.

12.4.3 Configuring data

Each data set has the form of the following example

```
<dataset name="grades" marks="dots" connected="no" stems="no">
  data
</dataset>
```

All of the arguments to the `dataset` element are optional. The `name`, if given, will appear in a legend at the upper right of the plot. The `marks` option can take one of the following values:

- `none`: (the default) No mark is drawn for each data point.
- `points`: A small point identifies each data point.
- `dots`: A larger circle identifies each data point.
- `various`: Each dataset is drawn with a unique identifying mark. There are 10 such marks, so they will be recycled after the first 10 data sets.
- `pixels`: A single pixel identified each data point.

The `connected` argument can take on the values “yes” and “no.” It determines whether successive datapoints are connected by a line. The default is that they are. Finally, the `stems` argument, which can also take on the values “yes” and “no,” specifies whether stems should be drawn. Stems are lines drawn from a plotted point down to the x axis. Plots with stems are often called “stem plots.”

The DTD is:

```
<!ELEMENT dataset (m | move | p | point)*>
    <!ATTLIST dataset connected (yes | no) #IMPLIED
          marks (none | dots | points | various | pixels) #IMPLIED
```

```
            name CDATA #IMPLIED
            stems (yes | no) #IMPLIED>
```

The default values of these arguments can be changed by preceding the `dataset` elements with a `default` element, as in the following example:

```
<default connected="no" marks="dots" stems="yes"/>
```

The DTD for this element is:

```
<!ELEMENT default EMPTY>
<!ATTLIST default connected (yes | no) "yes"
       marks (none | dots | points | various | pixels) "none"
       stems (yes | no) "no">
```

If the following element occurs:

```
<reuseDatasets/>
```

then datasets with the same name will be merged. This makes it easier to combine multiple data files that contain the same datasets into one file. By default, this capability is turned off, so datasets with the same name are not merged.

12.4.4 Specifying data

A dataset has the form

```
<dataset options>
  data
</dataset>
```

The data itself are given by a sequence of elements with one of the following forms:

```
<point y="yValue">
<point x="xValue" y="yValue">
<point y="yValue" lowErrorBar="low" highErrorBar="high">
<point x="xValue" y="yValue" lowErrorBar="low" highErrorBar="high">
```

To reduce file size somewhat, they can also be given as

```
<p y="yValue">
<p x="xValue" y="yValue">
<p y="yValue" lowErrorBar="low" highErrorBar="high">
<p x="xValue" y="yValue" lowErrorBar="low" highErrorBar="high">
```

The first form specifies only a Y value. The X value is implied (it is the count of points seen before in this data set). The second form gives both the X and Y values. The third and fourth forms give low and high error bar positions (error bars are use to indicate a range of values with one data point). Points

given using the syntax above will be connected by lines if the `connected` option has been given value "yes" (or if nothing has been said about it).

Data points may also be specified using one of the following forms:

```
<move y="yValue">
<move x="xValue" y="yValue">
<move y="yValue" lowErrorBar="low" highErrorBar="high">
<move x="xValue" y="yValue" lowErrorBar="low" highErrorBar="high">

<m y="yValue">
<m x="xValue" y="yValue">
<m y="yValue" lowErrorBar="low" highErrorBar="high">
<m x="xValue" y="yValue" lowErrorBar="low" highErrorBar="high">
```

This causes a break in connected points, if lines are being drawn between points. I.e., it overrides the `connected` option for the particular data point being specified, and prevents that point from being connected to the previous point.

12.4.5 Bar graphs

To create a bar graph, use:

```
<barGraph width="barWidth" offset="barOffset"/>
```

You will also probably want the `connected` option to have value "no." The *barWidth* is a real number specifying the width of the bars in the units of the X axis. The *barOffset* is a real number specifying how much the bar of the *i*-th data set is offset from the previous one. This allows bars to "peek out" from behind the ones in front. Note that the front-most data set will be the first one.

12.4.6 Histograms

To configure a histogram on a set of data, use

```
<bin width="binWidth" offset="binOffset"/>
```

The *binWidth* option gives the width of a histogram bin. I.e., all data values within one *binWidth* are counted together. The *binOffset* value is exactly like the *barOffset* option in bar graphs. It specifies by how much successive histograms "peek out."

Histograms work only on Y data; X data is ignored.

12.5 Old Textual File Format

Instances of the PlotBox and Plot classes can read a simple file format that specifies the data to be plotted. This file format predates the PlotML format, and is preserved primarily for backward compatibility. In addition, it is significantly more concise than the PlotML syntax, which can be advantageous, particularly in networked applications.

In this older syntax, each file contains a set of commands, one per line, that essentially duplicate

the methods of these classes. There are two sets of commands currently, those understood by the base class PlotBox, and those understood by the derived class Plot. Both classes ignore commands that they do not understand. In addition, both classes ignore lines that begin with "#", the comment character. The commands are not case sensitive.

12.5.1 Commands Configuring the Axes

The following commands are understood by the base class PlotBox. These commands can be placed in a file and then read via the read() method of PlotBox, or via a URL using the PlotApplet class. The recognized commands include:

- `TitleText:` *`string`*
- `XLabel:` *`string`*
- `YLabel:` *`string`*

These commands provide a title and labels for the X (horizontal) and Y (vertical) axes. A *`string`* is simply a sequence of characters, possibly including spaces. There is no need here to surround them with quotation marks, and in fact, if you do, the quotation marks will be included in the labels.

The ranges of the X and Y axes can be optionally given by commands like:

- `XRange:` *`min, max`*
- `YRange:` *`min, max`*

The arguments *`min`* and *`max`* are numbers, possibly including a sign and a decimal point. If they are not specified, then the ranges are computed automatically from the data and padded slightly so that datapoints are not plotted on the axes.

The tick marks for the axes are usually computed automatically from the ranges. Every attempt is made to choose reasonable positions for the tick marks regardless of the data ranges (powers of ten multiplied by 1, 2, or 5 are used). However, they can also be specified explicitly using commands like:

- `XTicks:` *`label position, label position, ...`*
- `YTicks:` *`label position, label position, ...`*

A *`label`* is a string that must be surrounded by quotation marks if it contains any spaces. A position is a number giving the location of the tick mark along the axis. For example, a horizontal axis for a frequency domain plot might have tick marks as follows:

```
XTicks: -PI -3.14159, -PI/2 -1.570795, 0 0, PI/2 1.570795, PI 3.14159
```

Tick marks could also denote years, months, days of the week, etc.

The X and Y axes can use a logarithmic scale with the following commands:

- `XLog: on`
- `YLog: on`

The tick labels, if computed automatically, represent powers of 10. The log axis facility has a number of limitations, which are documented in "Limitations" on page 12-24.

By default, tick marks are connected by a light grey background grid. This grid can be turned off with the following command:

- `Grid: off`

It can be turned back on with

- `Grid: on`

Also, by default, the first ten data sets are shown each in a unique color. The use of color can be turned off with the command:

- `Color: off`

It can be turned back on with

- `Color: on`

Finally, the rather specialized command

- `Wrap: on`

enables wrapping of the X (horizontal) axis, which means that if a point is added with X out of range, its X value will be modified modulo the range so that it lies in range. This command only has an effect if the X range has been set explicitly. It is designed specifically to support oscilloscope-like behavior, where the X value of points is increasing, but the display wraps it around to left. A point that lands on the right edge of the X range is repeated on the left edge to give a better sense of continuity. The feature works best when points do land precisely on the edge, and are plotted from left to right, increasing in X.

All of the above commands can also be invoked directly by calling the corresponding public methods from some Java code.

12.5.2 Commands for Plotting Data

The set of commands understood by the Plot class support specification of data to be plotted and control over how the data is shown.

The style of marks used to denote a data point is defined by one of the following commands:

- `Marks: none`
- `Marks: points`
- `Marks: dots`
- `Marks: various`
- `Marks: pixels`

Here, `points` are small dots, while `dots` are larger. If `various` is specified, then unique marks are used for the first ten data sets, and then recycled. If `pixels` is specified, then a single pixel is drawn. Using no marks is useful when lines connect the points in a plot, which is done by default. If the above directive appears before any DataSet directive, then it specifies the default for all data sets. If it appears after a DataSet directive, then it applies only to that data set.

To disable connecting lines, use:

- `Lines: off`

To re-enable them, use

- `Lines: on`

You can also specify “impulses”, which are lines drawn from a plotted point down to the x axis. Plots with impulses are often called “stem plots.” These are off by default, but can be turned on with the command:

- `Impulses: on`

or back off with the command

- `Impulses: off`

If that command appears before any DataSet directive, then the command applies to all data sets. Otherwise, it applies only to the current data set.

To create a bar graph, turn off lines and use any of the following commands:

- `Bars: on`
- `Bars: width`
- `Bars: width, offset`

The *`width`* is a real number specifying the width of the bars in the units of the x axis. The *`offset`* is a real number specifying how much the bar of the i-th data set is offset from the previous one. This allows bars to "peek out" from behind the ones in front. Note that the front-most data set will be the first one. To turn off bars, use

- `Bars: off`

To specify data to be plotted, start a data set with the following command:

- `DataSet: string`

Here, *`string`* is a label that will appear in the legend. It is not necessary to enclose the string in quotation marks.

To start a new dataset without giving it a name, use:

- `DataSet:`

In this case, no item will appear in the legend.

If the following directive occurs:

- `ReuseDataSets: on`

then datasets with the same name will be merged. This makes it easier to combine multiple data files that contain the same datasets into one file. By default, this capability is turned off, so datasets with the same name are not merged.

The data itself is given by a sequence of commands with one of the following forms:

- *`x, y`*
- `draw:` *`x, y`*
- `move:` *`x, y`*
- *`x, y, yLowErrorBar, yHighErrorBar`*
- `draw:` *`x, y, yLowErrorBar, yHighErrorBar`*
- `move:` *`x, y, yLowErrorBar, yHighErrorBar`*

The `draw` command is optional, so the first two forms are equivalent. The `move` command causes a break in connected points, if lines are being drawn between points. The numbers *`x`* and *`y`* are arbitrary numbers as supported by the Double parser in Java (e.g. "1.2", "6.39e-15", etc.). If there are four numbers, then the last two numbers are assumed to be the lower and upper values for error bars. The numbers can be separated by commas, spaces or tabs.

12.6 Compatibility

Figure 12.10 shows a small set of classes in the compat package that support an older ascii and binary file formats used by the popular `pxgraph` program (an extension of `xgraph` to support binary formats). The PxgraphApplication class can be invoked by the pxgraph executable in $PTII/bin. See the PxgraphParser class documentation for information about the file format.

12.7 Limitations

The plot package is a starting point, with a number of significant limitations.

- A binary file format that includes plot format information is needed. This should be an extension of PlotML, where an external entity is referenced.
- If you zoom in far enough, the plot becomes unreliable. In particular, if the total extent of the plot is more than 2^{32} times extent of the visible area, quantization errors can result in displaying points or lines. Note that 2^{32} is over 4 billion.
- The log axis facility has a number of limitations. Note that if a logarithmic scale is used, then the values must be positive. **Non-positive values will be silently dropped**. Further log axis limitations are listed in the documentation of the _gridInit() method in the PlotBox class.
- Graphs cannot be currently copied via the clipboard.
- There is no mechanism for customizing the colors used in a plot.

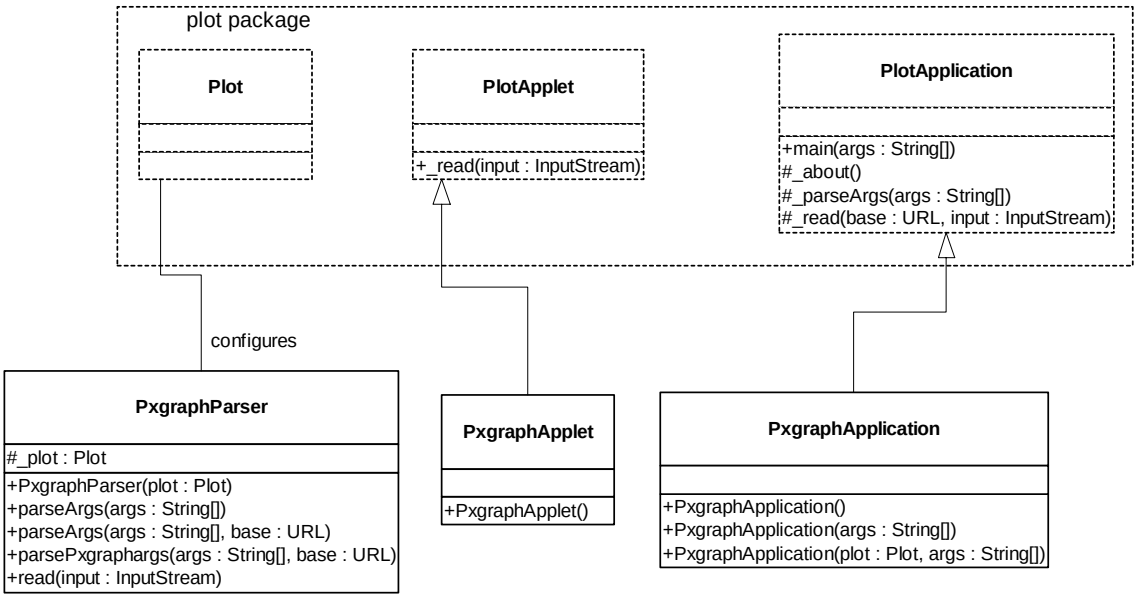

FIGURE 12.10. The compat package provides compatibility with the older pxgraph program.