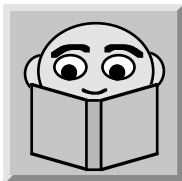


17. Applying the ABCs -- Graphical Display of Documents

Overview

So far in this book we've presented three complete example programs with user interfaces. In Chapters 15 and 16, we saw how individual user interface elements could be pieced together to form a working model of a hand-held electronic calculator. In one of the Calculator programs, the workings of a hand-held calculator were encapsulated in just a single **Calculator** class added to the Application Builder Classes code. Way back in the first chapter, we saw how easy it was to add a *document* to a program. In the address book program presented in that chapter, the ABCs gave us the ability to present data in textual form in a window, store it in a file or print it out on a page. In both of these programs, the Application Builder Classes did most of the work for us. This is the way programming is supposed to be!

But there's still one thing that's not quite right. Our wonderful "*graphical* user interfaces" haven't been completely graphical. In this final chapter on user interfaces, we will fill in this one remaining gap in our discussion of user interfaces (and unfortunately, a gap in the Prograph CPX manuals as well) -- presenting *graphical* information to the user in a window. Rather than displaying the data of a document as just text, we'll *draw* the data into the window. The program we'll discuss here will take Prograph full circle -- using a visual language and visual tools to provide a fully *visual* interface for applications. Even in this more complicated program, we'll see how the Application Builder Classes and Editors remove most of the programming effort.



**For More
Information...**

In this chapter, we will access fairly generic operating system drawing routines that should be similar for all platforms that run Prograph. It is beyond the scope of this book to delve into the operating system of the computer; in fact, the purpose of this chapter is to demonstrate how Prograph CPX shields you from having to call most low-level routines. However, if you do need to produce more complex graphics, consult the programming documentation for your computer of choice.

The Graphical Data Plotter Program

Early computers were hindered by being restricted to displaying only textual information. When the first graphical user interfaces were introduced, a whole new world was opened up to both the computer user and computer programmer. Not only could you describe your data, but you could also *draw* it on the screen. Now we are going to take that bold step ourselves and show you how to draw the data stored in a document into a window and then print it on paper. Most importantly, we think that you'll be amazed at how easy Prograph will make this programming exercise -- by taking advantage of the

Application Builder Classes' **View** class, even graphics programming becomes child's play!

Our project is a program that will plot data in a window, complete with plot axes, axis labels and a plot title. An example document window is shown in Figure 17.1. Here we've plotted some laboratory data, a physiological recording called an electroretinogram. However, the code of this program could be just as easily be used to plot other data such as business graphs. Notice that the plot's X-axis and Y-axis are automatically given the appropriate range of values to plot the data correctly. All of this is handled by the Waveform Plotter program's code.

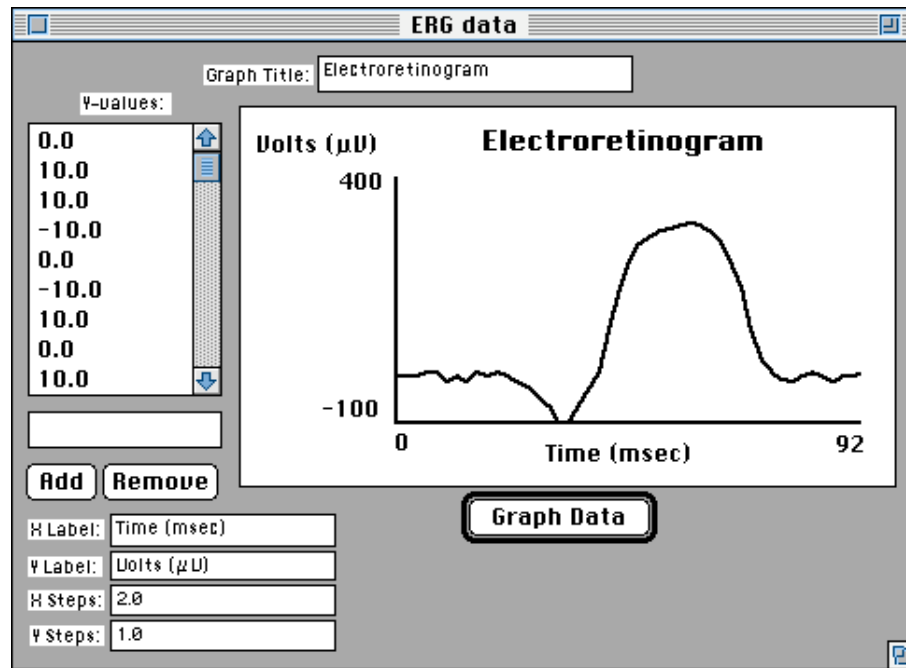


Figure 17.1: The Data Plotter document window with sample data stored in the 'ERG data' file

As you might expect, most of the classes used in this program are either already in the Application Builder Classes or will be created for us by the Application Builder Editors. Figure 17.2 shows some of the subclasses that will be housed in the Data Plotter Workspace section (we'll create this section when we open the ABC Starter Project). All except the **Waveform Data** subclass are made for us -- we will have to subclass **Waveform Data** by ourselves.

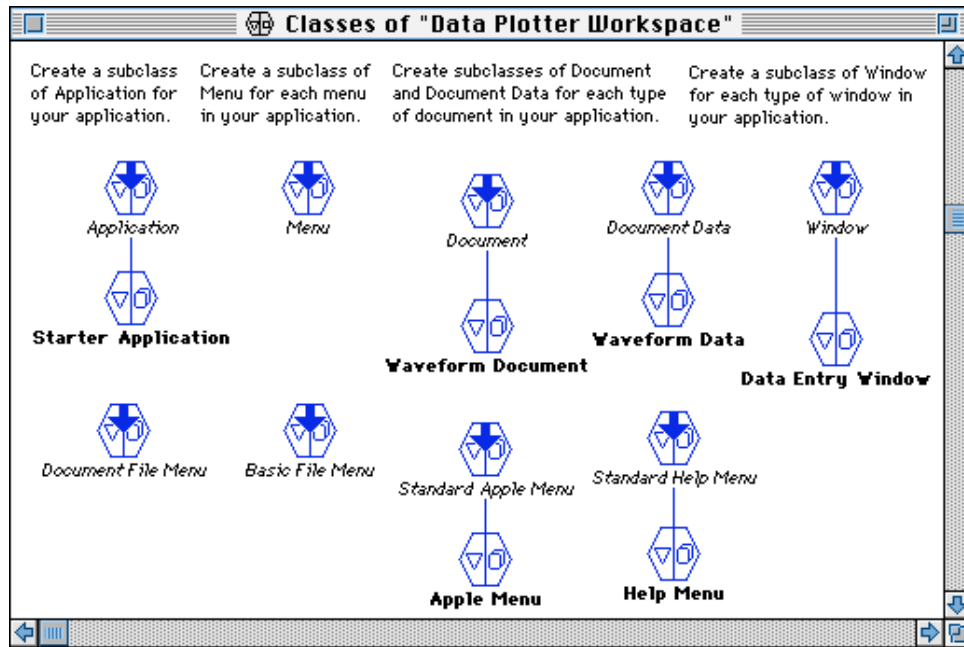


Figure 17.2: The subclasses of the Data Plotter Workspace section

The program will also use a custom subclass of one of the ABCs to display the graphical representation of the document data. The **View** class, a subclass of **Window Item**, is one of the ABC's most general-purpose classes. It is used mostly to contain and manage other user interface elements. For example, all windows contain a **View** in which all of its elements are placed. In addition, **Views** may contain other **Views**, which could be considered to be subviews.

But there's another feature of **Views** that is less obvious. The **View** class, like other subclasses of **Window Item**, can contain *graphics*. This means that we can draw whatever we want into a **View**. Rather than add new drawing actions to the **View** class directly, we'll be adding them to a new **Waveform View** subclass shortly.

Let's dig in and start building the program. Open the ABC Starter Project and save the new project under the name Data Plotter Project, and the Workspace section as Data Plotter Workspace. Once again, by including the ABCs in our program, most of the work needed to manage a user interface has been done for us.

Add a new section entitled **Waveform View** to the project. Open the section's Classes window and create a new class. Transform this class into an *alias* to the **View** class. Now create a subclass of **View** and name it **Waveform View** (see Figure 17.3). At this point, you may also create the **Waveform Data** subclass of **Document Data** in the Data Plotter Workspace section.

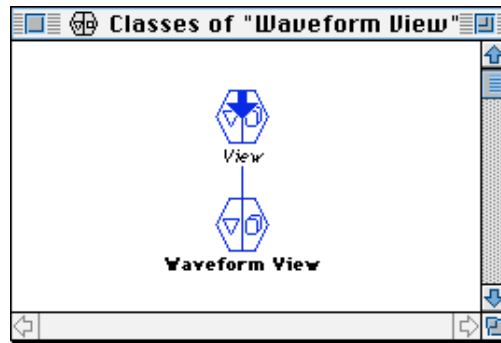


Figure 17.3: The Waveform View subclass in the Waveform View section

Open the Application Editor and enter the name and signature of the application as shown in Figure 17.4.

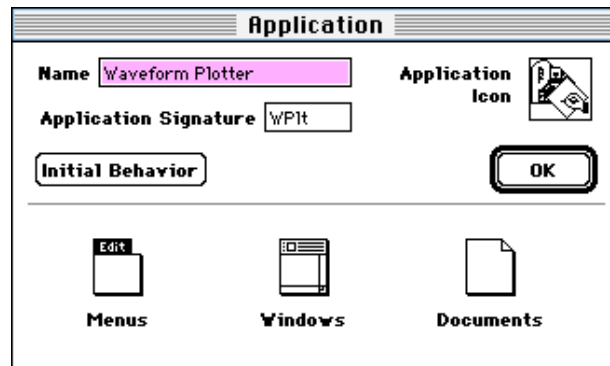


Figure 17.4: Defining the Application with the Application Editor

The menus for the program, as specified in the Menubar Editor, are shown in Figure 17.5. We will use the Document File menu supplied as a predefined menu in the Menu Editor's list of available menus. No other menus are needed.

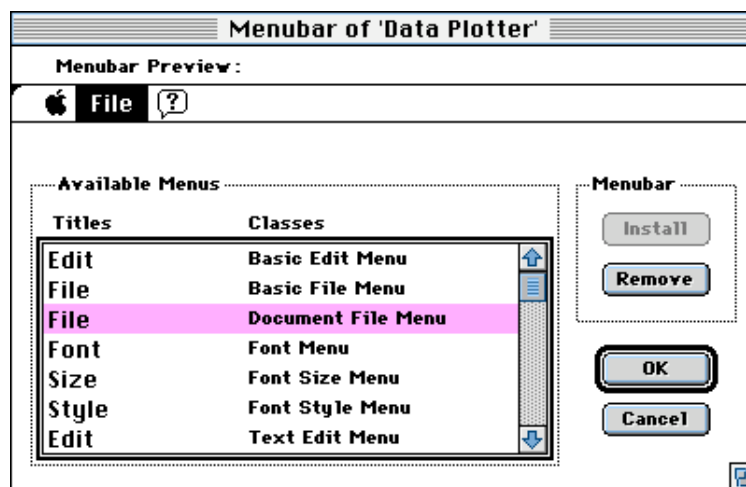
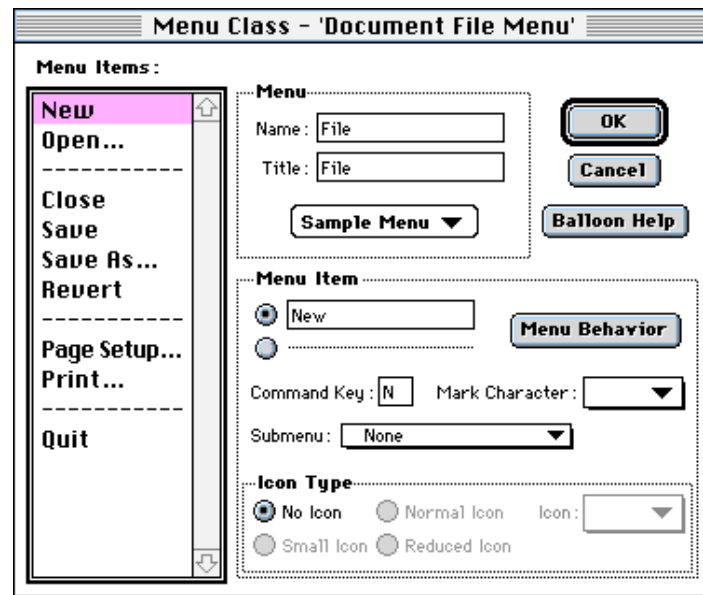


Figure 17.5: Defining the menus of the Data Plotter program

The Document File menu (Figure 17.6) adds a lot of functionality to our program. It gives the program the ability to create a new document, open a previously-saved document, save the current document, print the document's data on paper and exit the program by selecting a menu item. All of the behaviors of these menu items are already defined for us -- we now have menus for our program without any effort at all!

**Figure 17.6: The Document File menu**

We'll include only one window in the Waveform Plotter program. This window will display the plot of our data. But before we can create the window, we need to add the **Waveform View** subclass of the class to the View Editor's View palette, just like we added new **Push Button** subclasses to the Controls palette in Chapter 15. Otherwise, we won't be able to add our custom **Waveform View** user interface object directly to the window.

First select the Preferences... menu item to open the Preferences dialog box and uncheck the Hide Interpreter Only check box. This will make the ABE sections of the ABCs visible in the Sections window. Open the **View Editor Preferences** class of the "•Editor Preferences" section. Edit the **Floater Contents** attribute's third sublist (the one with the Views in it) by adding our new **Waveform View** subclass to its end, as shown in Figure 17.7. Now the View palette will recognize our new **Waveform View** type.

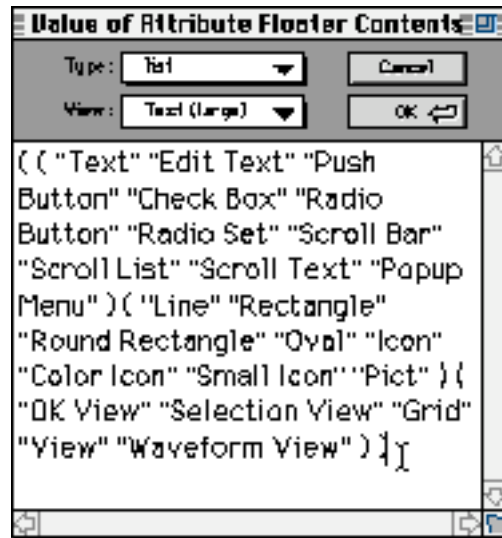


Figure 17.7: Edited Floater Contents attribute of the View Editor Preferences class with new Waveform View subclass added

We also must tell the View palette to display an icon for the new **Waveform View** subclass so that this new type of View that may be dragged into a window's view. Open the **Floater Picts** attribute of the **View Editor Preferences** class and duplicate the View's picture resource number at the end of the third list (for the View palette) for the new **Waveform View** as well (see the duplicated number 37 in the third list in Figure 17.8).

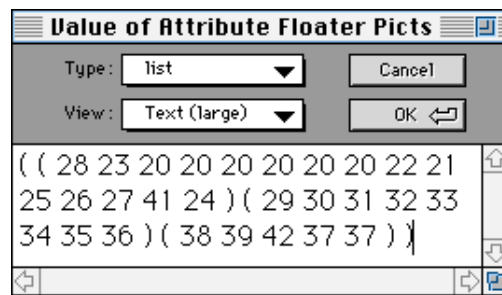


Figure 17.8: Floater Picts attribute of the View Editor Preferences class

The View palette should now contain our new **Waveform View** subclass and resemble Figure 17.9. We can now drag a **Waveform View** from the View palette to the Waveform window and it will automatically have all of the new actions that we'll add to the **Waveform View** as well as those of a **View**.

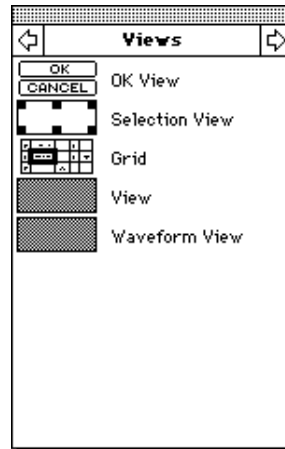


Figure 17.9: Views palette with new Waveform View subclass added

Now we can create the window that will display our document data. Select the Create Window menu item and when the window name dialog box appears, give the window the name Waveform Window, which will also automatically create a Waveform Window subclass of the Window class (see Figure 17.10).

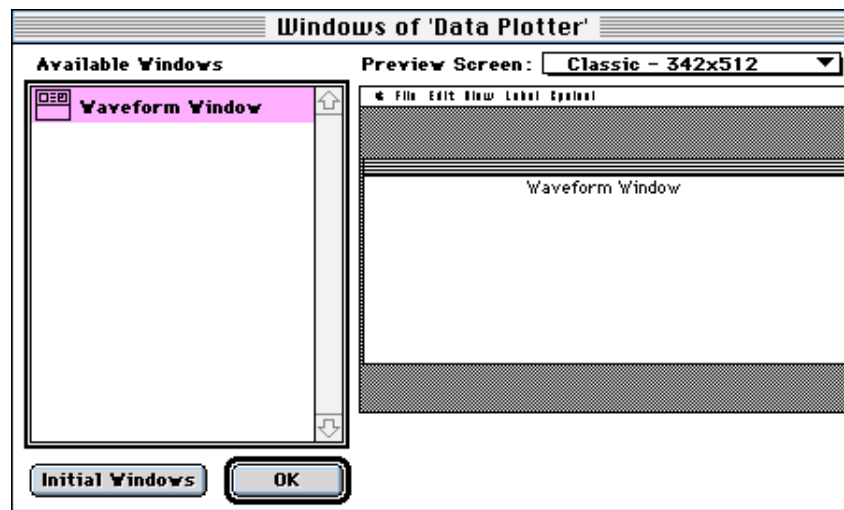


Figure 17.10: Creating the Waveform Window

Open the View Editor for the new window and fill it in as shown below in Figure 17.11. Give the window a gray background with the Background Editor. The central large element of the window is our Waveform View. The majority of the elements of the window are Text, Edit Text and Push Button objects. Name the Edit Text elements the following names (as suggested by their labelling text in the window): 'Graph Title Edit Text', 'X-axis Label Edit Text', 'Y-axis Label Edit Text', 'X-axis Steps Edit Text', and 'X-axis Steps Edit Text'. The Edit Text object below the scrolling list should be named 'Y-value Edit Text'. The Push Button elements should be named 'Add To List Button', 'Remove From List Button' and 'Graph Data Button'. The Graph Data button should also

be made the default button of the window (the Push Button that's selected by pressing the Return key of the keyboard as well as by directly selecting it with the mouse) with the Default Button menu item of the View menu.



Figure 17.11: Defining the Main View of the Waveform Window

The central element in the window is a large View object. We'll discuss this element in detail soon. Finally, there is a scrolling list element on the left side of the window. The settings for the Scroll List in the Data Plotter window are shown in the Scroll List Editor of Figure 17.12. Initially, the Item List of the Scroll List, that is, the list of members of the list displayed in this user interface element, is empty.

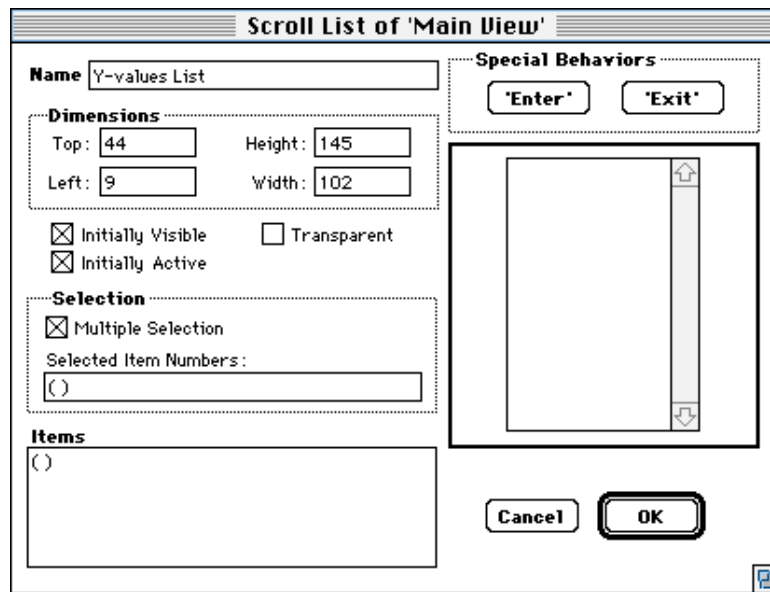


Figure 17.12: Defining the Scroll List of the Waveform Window

Although we've defined several Edit Text text-editing boxes in which users may type, we have not defined how the user may move from one to the next. The tab key on the keyboard is usually used to move the cursor from one Edit Text to another, deactivating the previous Edit Text and activating the next. There is nothing present yet in our program that determines the order in which the Edit Texts of our window will be accessed with the tab key. In addition, you should be aware that the tab key also activates or deactivates the scrolling list. To set the order in which the tab key activates window items, we select the Show Tab Order menu item to graphically depict and modify the tab order. Redefine the tab order to look like that of Figure 17.13, where the graph title text-editing box is accessed first, then the data entry text-editing box under the scroll list, then the graph label and data point step text-editing boxes in order, and finally the scrolling list itself.

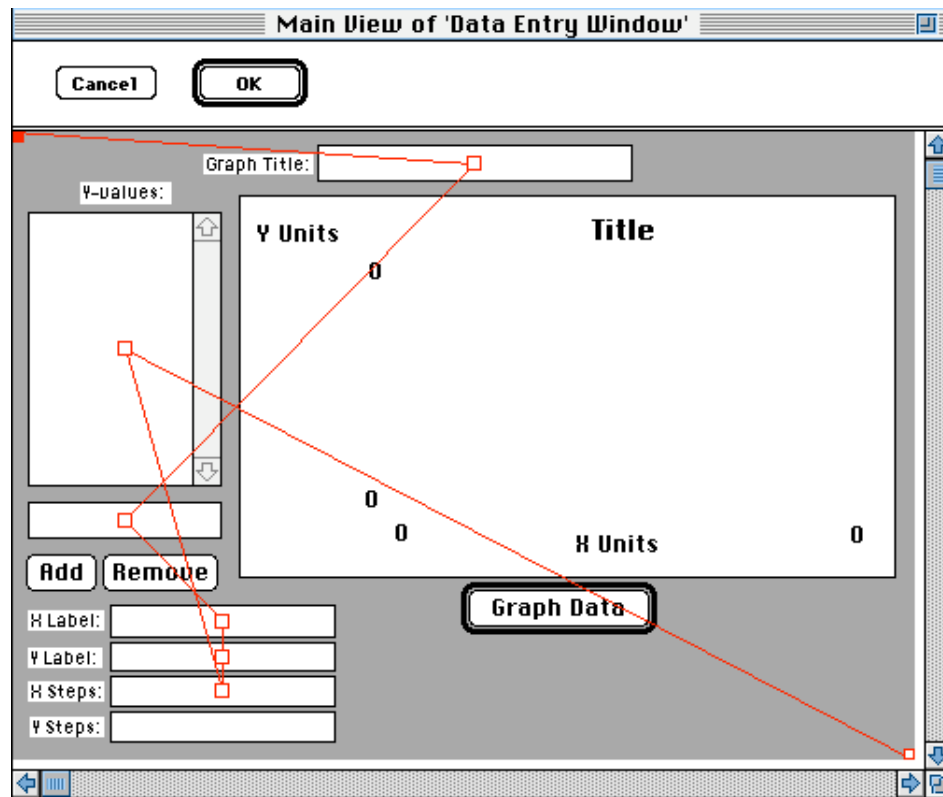


Figure 17.13: Defining the tab order of Edit Text and Scroll List objects in the Waveform Window

Most of the objects in the window don't need behaviors, but the three Push Buttons do. Define the Click Behavior of the Add To List button as shown in Figure 17.14. Clicking on this button will call a method named **Add Data**, which resides in the **Waveform Document** class (as revealed by the first input to the method -- The Document). The second input to the method is Y-values List, the **Scroll List** object that will hold the list of data points to be plotted, and the final input is the value currently held by the Y-value Edit Text, that is, the value of the new data point to be added to the data point list.

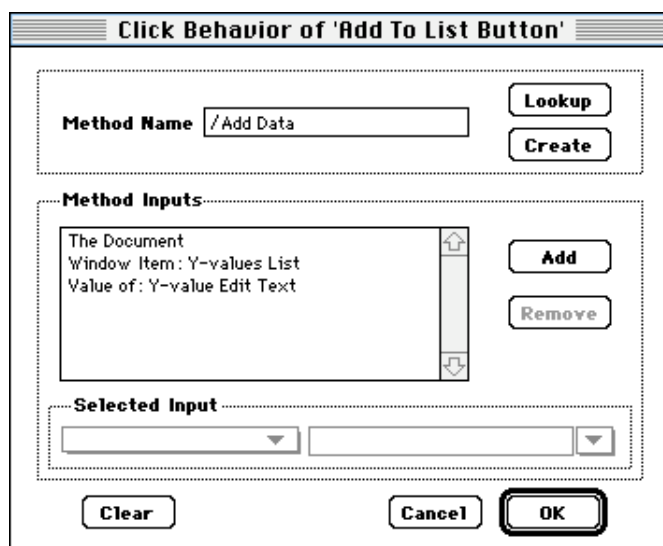


Figure 17.14: The Click Behavior of the Add To List Push Button

The Click Behavior of the Remove From List button is seen in Figure 17.15. Clicking on this button will call the **Remove Data** method, which also resides in the **Waveform Document** class. The second input to that method, once again, is the **Scroll List** named **Y-values List**. You may have noticed in the Scroll List editor shown in Figure 17.12 that the **Multiple Selection** check box was selected. This means that the user will be able to select multiple data point values in the scrolling list and remove all of them at once by clicking on the **Remove From List** button.

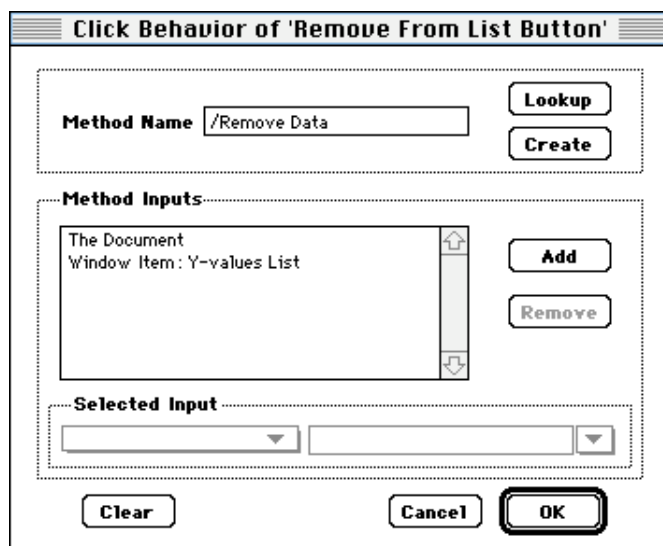


Figure 17.15: The Click Behavior of the Remove From List Push Button

Finally, the Click Behavior of the last button, the **Graph Data** button, is shown in Figure 17.16. This is the most important button of the window, since it causes the plot of the waveform data to graph itself. The button calls a method named **Draw** in the

Waveform Document class. The second input of this method is the large View object in the window, which we'll name 'Plot View'. Therefore, you'll have to wait until we define this view (see the next two figures) before you set this click behavior.

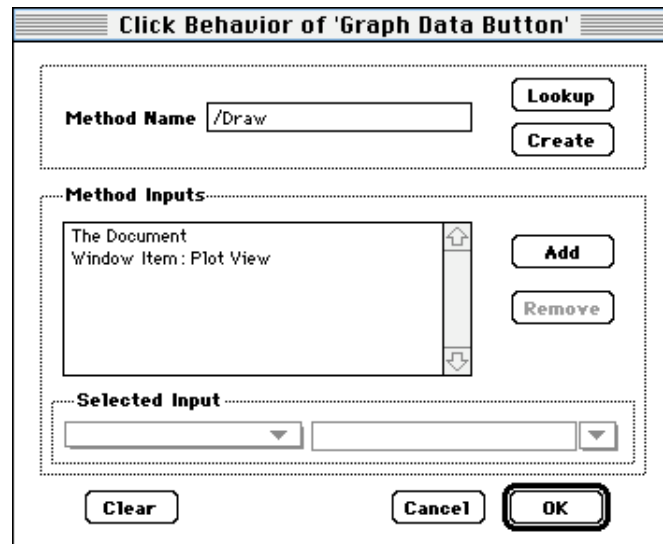


Figure 17.16: The Click Behavior of the Graph Data Push Button

It's time to define the View in which the waveform plot will be graphed. If you double-click within the rectangular bounds of the View, a Subview Editor dialog will appear (Figure 17.17). Why is it a *subview* rather than a view? This is because all windows by default already contain one view -- the 'Main View' of the window. Any view we add to the window is necessarily a *subview* of the Main View. This is pointed out by the title of the Subview Editor dialog -- 'Subview of 'Main View''. The specifications for this subview are shown in Figure 17.18. The subview is named 'Plot View' and it is both visible and active. The check box labelled 'Transparent' in the editor is left unchecked, meaning that the contents of this subview will be drawn on top of the underlying Main View and cover up that view rather than let the underlying view show through.

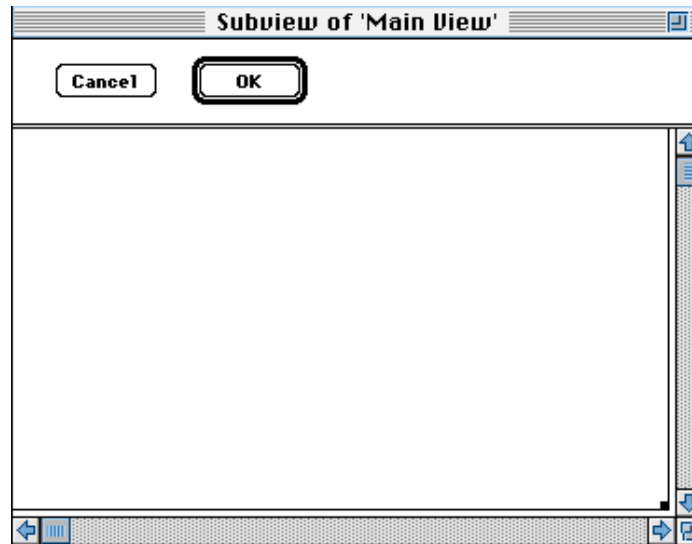


Figure 17.17: The SubView Editor

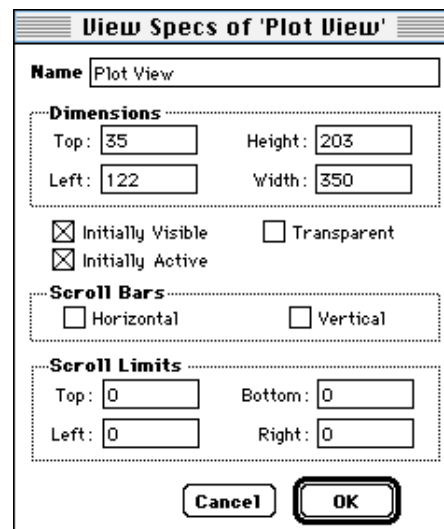


Figure 17.18: The View Specs Editor showing the specifications of the 'Plot View' Subview

At this stage in designing the program, we have a decision to make -- we have two possible ways in which how the waveform graph display may be defined. The easiest way would be to draw the graph and its labels *directly* into the Plot View with graphics commands supplied by the operating system. This is an acceptable and desirable approach most of the time, but it has two less obvious shortcomings. First, the programmer must have intimate knowledge of graphics, font-handling and text-drawing calls to the operating system for his or her specific computer. Since this book stresses the platform-independent programming capabilities of Prograph CPX, we will avoid the very platform-specific font and text rendering calls. Secondly, drawing a line graph of the

waveform data directly into the subview will restrict our flexibility in how we may present data.

We will choose a slightly more complicated means of plotting the data into a view. Rather than draw directly into the Plot View, we will enclose another subview *within* it. This subview will be an instance of the **Waveform View** class that we created earlier in this chapter (see the selected rectangular view in the Plot View's editor dialog in Figure 17.19). We'll also place labels for the data plot within the Plot View. Why did we include this extra level of nested views when we could have drawn directly into the Plot View? It allows us to present the data in many different ways easily. Every **View** object has an attribute named **Visible?** that determines whether or not the view is displayed to the user. We could actually place *two* graphing subviews within the Plot View -- one **Waveform View** subview (a subclass of **View**) and a second subview based upon another subclass of **View** that could present the waveform data in another way, such as a scatter plot or bar graph. By making one of these graphing subviews invisible and the other visible, the program could switch back and forth between which graph format it displayed in the window, yet maintain the correct labels on the graph axes. We could then let the user select the type of graph display they would prefer to view and print.

This is an important point since, at present, Prograph CPX is limited to allow only one display window per document. If we could present several document windows, we could simply display one window for entering data (for example, into a spreadsheet-like grid) and separate graph windows for each different way of displaying the data within the document. This is the way that most commercial data graphing programs work. But since we are limited to one window alone, we must both enter data and display a graph in the same window. If we want to display different types of graphs, we must do so within the confines of this same window. The easiest way to do so is to swap views in and out of the window by making them visible or invisible.

While our example program lays the groundwork for multiple graph displays, it does not actually do so due to page number limitations. We will leave that as an exercise for the reader.

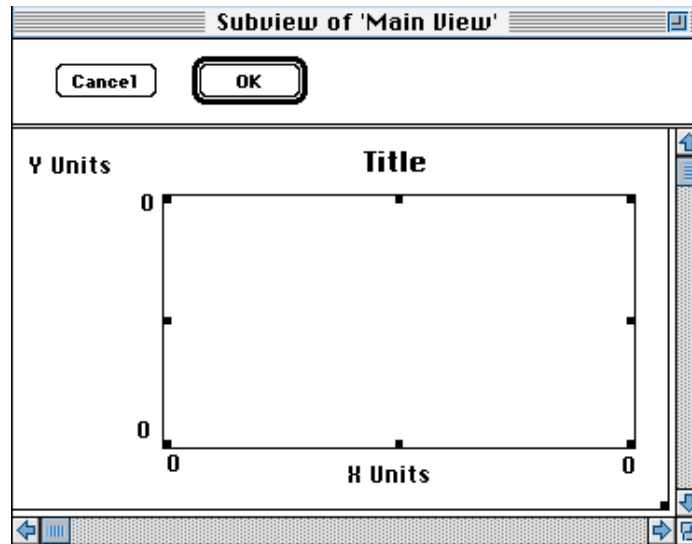


Figure 17.19: The 'Plot View' subview showing the inclusion of its text labels and an embedded Waveform View subview

When the graph is presented, we don't want the graph to have a box drawn around it, like all Views do by default. Highlight the Plot View, then select the Border... menu item to enter the Border Editor (Figure 17.20). The default border around a View is a black border that is one pixel wide.

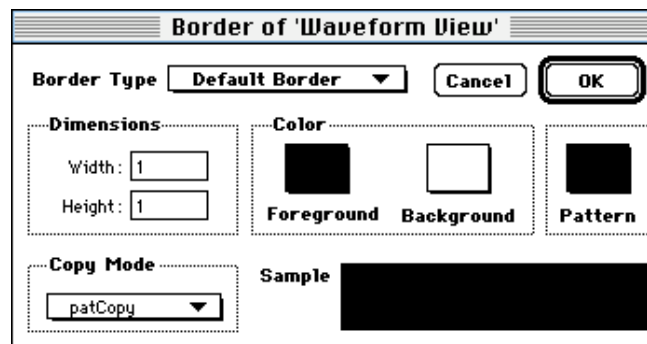


Figure 17.20: The Border Editor

Change the border type to 'None'. This will prevent a border being drawn around the Waveform View (see Figure 17.21).

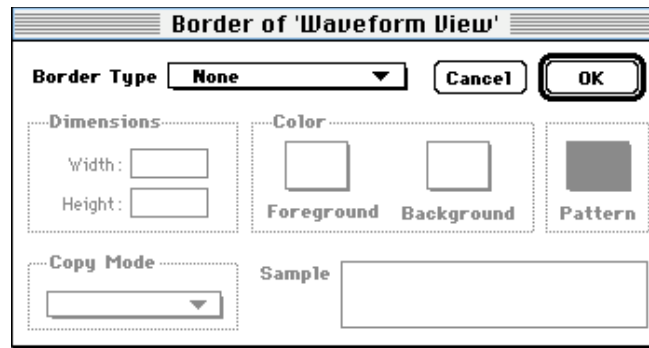


Figure 17.21: The 'Plot View' subview showing the inclusion of its text labels and an embedded Waveform View subview

The graph title and axis labels are simply Text objects. Name these labels as follows: Y-axis Label and X-axis Label for the units of the Y- and X-axes, Y Minimum Label and Y Maximum Label for the numbers on the Y-axis (and the corresponding X Minimum Label and X Maximum Label for the X-axis), and Data Type Label for the title of the plot. Name the Waveform View 'Waveform View'.

Most of the behind-the-scenes work of the program will be accomplished by the application's document. Let's define that document now. Enter the Documents Editor and create a new document named Waveform Document (see Figure 17.22). Install that document type as the Main Document for our application.

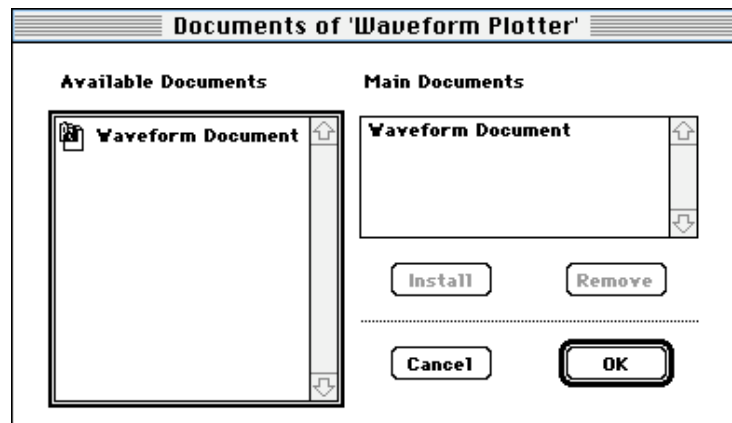


Figure 17.22: Creating the Waveform Document

Complete the specifications for the Waveform Document as shown in Figure 17.23. The document will access the data of the Waveform Data class. It will display its data in the Data Entry Window, and use the Print Layout class to produce hardcopies of the data plot. To save the data in a file, we'll simply store all of the attributes of the Waveform Data object that we subclassed from Document Data into the document file with the help of the Object File class.

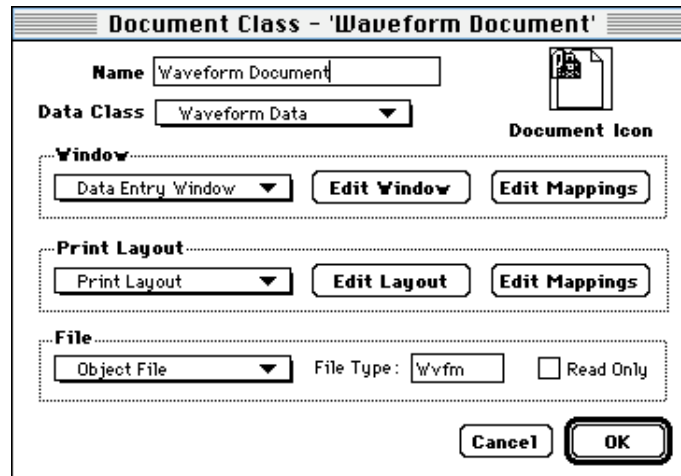


Figure 17.23: Defining the Waveform Document

Unlike the document used in the Address Book program of Chapter 1, the current program does not need any *mapping* of data to the window or to the printed page. This is because we are not just outputting text that the Document and Mapping classes can display for us -- we must *draw* the data ourselves into the Waveform View. Therefore the Waveform Document is pretty much complete at this point, except for one thing. The Print Layout class must still know how to print out the data shown in the Waveform Window.

Click on the Edit Layout button in the Document Editor to enter the Print Layout Editor (Figure 17.24). We'll name the layout 'Plot Layout'. Define the Page Type as "Print View", and tell it to use the window's Plot View (selected from the pop-up menu labelled 'Use Window's View:') as the contents of the Print View. This means that when we select the Print menu item as the program is executing, the contents of the Plot View, just as they appear on the screen, will be printed out as hardcopy. The check box labelled Resize Contents with Page scales the size of the printout to the size of the paper on which we print.

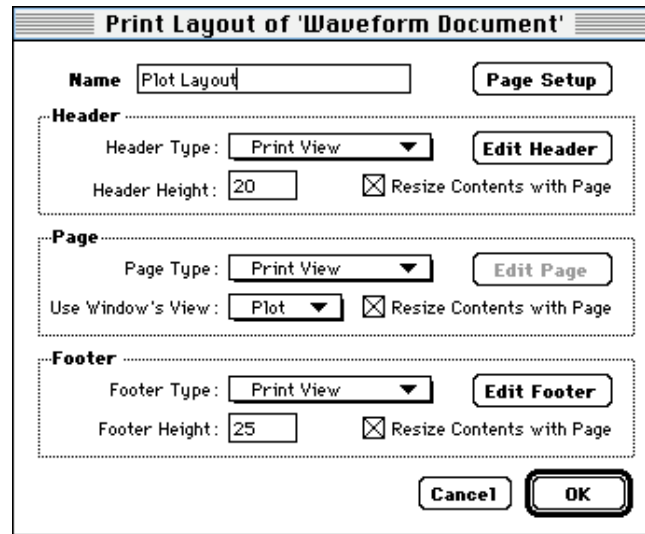


Figure 17.24: Defining the Waveform Document

Notice that we will also add a header and a footer to the printout. We use a Print View to print each of these as well. By clicking on the Edit Header button, the Header Editor is entered. This editor is much like the View editor, except that it allows the inclusion of special printing information by means of the Printing palette, shown in Figure 17.25. This includes the current date and time, as well as the page number of the printout, especially useful for multipage printouts.

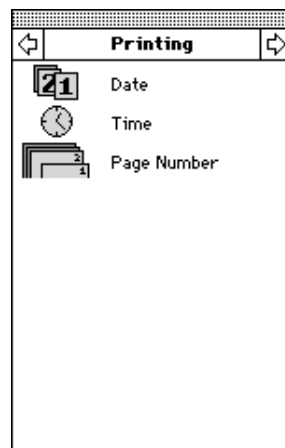


Figure 17.25: The Printing palette

Place Date and Time objects into the Header View as shown in Figure 17.26. When our document is printed onto a page, the current date and time will appear on the top of the page as well.

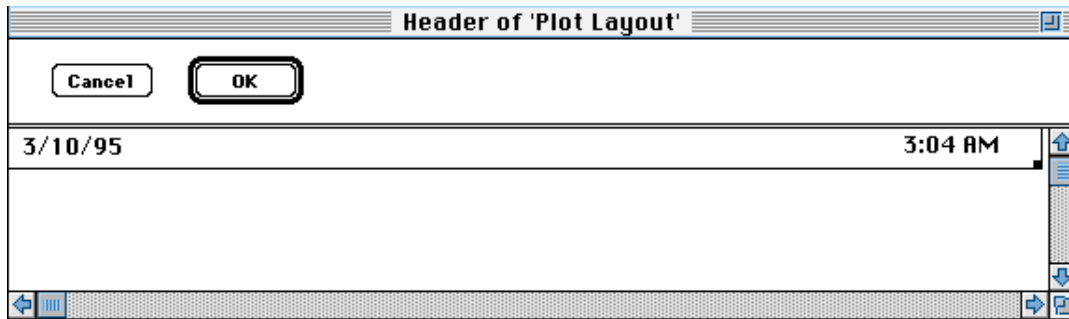


Figure 17.26: The Header Editor

Similarly, place a Page Number object into the Footer View (see Figure 17.27) to have the page number appear on the printed document page.

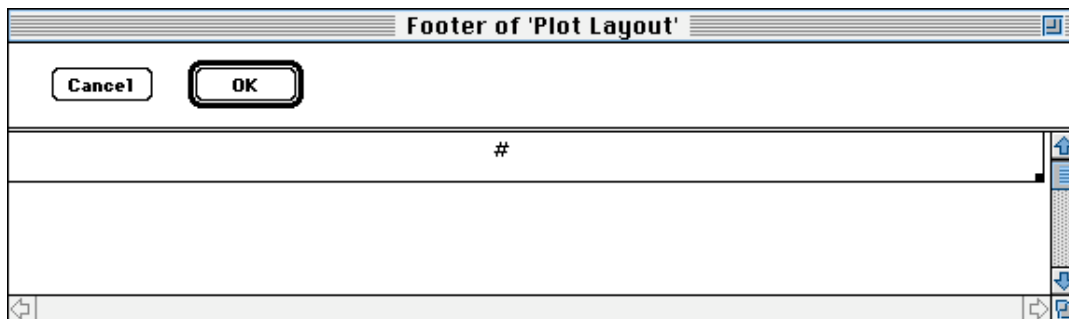


Figure 17.27: The Footer Editor

Writing the Code for the Data Plotter Program

Now that the user interface is prepared, it's time to start coding. The Waveform Plotter program will require the interaction of five classes that we must write methods for -- Waveform Document, Waveform Data, Waveform View, Data Entry Window and List Items Mapping. As their names suggest, Waveform Document and Waveform Data are responsible for storing and manipulating the data to be plotted, while Waveform View and Data Entry Window handle the display of that data on-screen. We will also add a utility class named List Items Mapping that will serve to get a list of data point values into and out of the scrolling list in the Data Entry window.

Specifically, the Waveform Data class will contain methods for finding the range of values that the data points hold, as well as a method for setting new values for the document data. Waveform Document will contain methods for adding data to the waveform, removing data, and initiating data plotting. The bulk of the work of this program will be done by methods of the Waveform View class, which will actually preparing the labels and plotting scale of the graph -- that is, the correspondence between a data point's value and its position within the graph -- and plot the data within the Waveform View in the Waveform Window or the Print Layout. The Data Entry

Window class will contain only one method to store new strings in the Text objects that make up the labels of the plot axes and the plot title.

The document data class of the ABCs uses specific methods to get and set their relevant attributes, named **Get Value** and **Set Value**. Likewise, the **View** class contains the **Set Value**, **Print** and **Draw** methods to set up a drawing, draw graphics into the View and print the contents of a View. In our **Waveform Data** and **Waveform View** subclasses of these ABC classes, we must overshadow these methods to perform the specific actions we desire from our waveform graph display and its representation in a file.

We'll start by designing the classes that will hold the document's data and plot it, beginning with the **Waveform Data** class, which we have already subclassed from **Document Data**. This class holds not only the actual data to be plotted, but also some extra attributes for labeling the plot and calculating the upper and lower limits of the plot for scaling its size into the Waveform View (see Figure 17.28).

The **data** attribute is a list of numbers to be plotted. These numbers correspond to the individual Y-axis values of the data plot. The **range**, **maximum** and **minimum** attributes are used to scale the limits of the Y-axis of the plot. The following two attributes, **ΔX** and **ΔY** , determine the units of measurement of the plot -- for example, in the electroretinogram data to be plotted, each Y-axis unit is 1 microvolt, while each X-axis unit is 2 milliseconds. **Current point** is a counter that we use to mark our current place when we performing calculations that involve the entire data list. The final three attributes -- Y-axis units, X-axis units and graph title are labels for the Y-axis, X-axis and the data plot title, respectively.

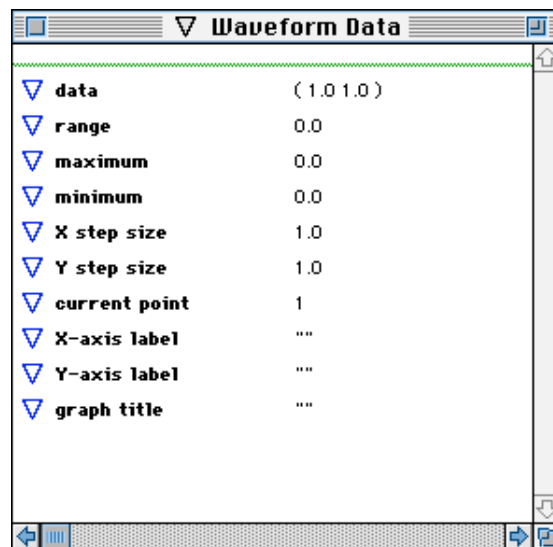


Figure 17.28: Attributes of the Waveform Data class

Waveform Data has only six class methods (Figures 17.29-17.35). Its initialization method (see Figure 17.29) sets the **data** attribute to an empty list and **current point** to 1, and zeros the **range**, **minimum** and **maximum** attributes.

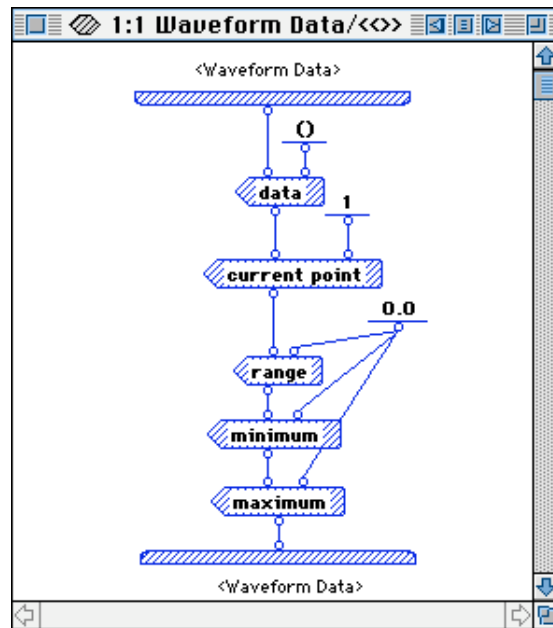


Figure 17.29: Initialization method of the Waveform Data class

The Find Maximum method (see Figure 17.30) returns the highest value in the document data found in the list of the **data** attribute. After setting the **current point** to 1, it ensures that **maximum** is assigned the lowest possible expected value of the data points. A list multiplex consisting of the Get max local method is then entered, which resets **maximum** to the maximum data point value.

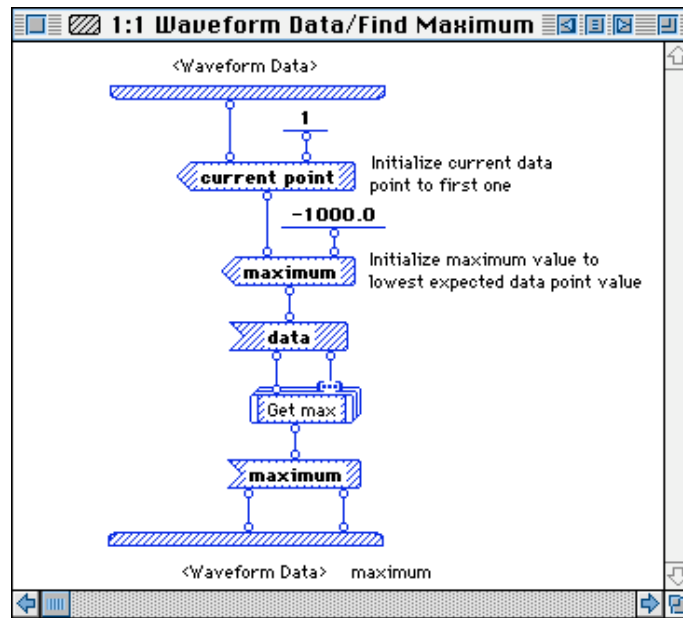


Figure 17.30: Find Maximum method of the Waveform Data class

The actual chore of finding the maximum value is carried out by the Get max local method (Figure 17.31), which tests each point in turn to see if it's higher than the current value of **maximum**. If not, **current point** is incremented (so that the next data point will be worked on during the next iteration through this local method) but **maximum** is left unchanged. If the current data point is higher than the previously-set maximum, **maximum** is reset to the new data point's value. After looping through the entire list, **maximum** will be set to the highest level of all of the data points.

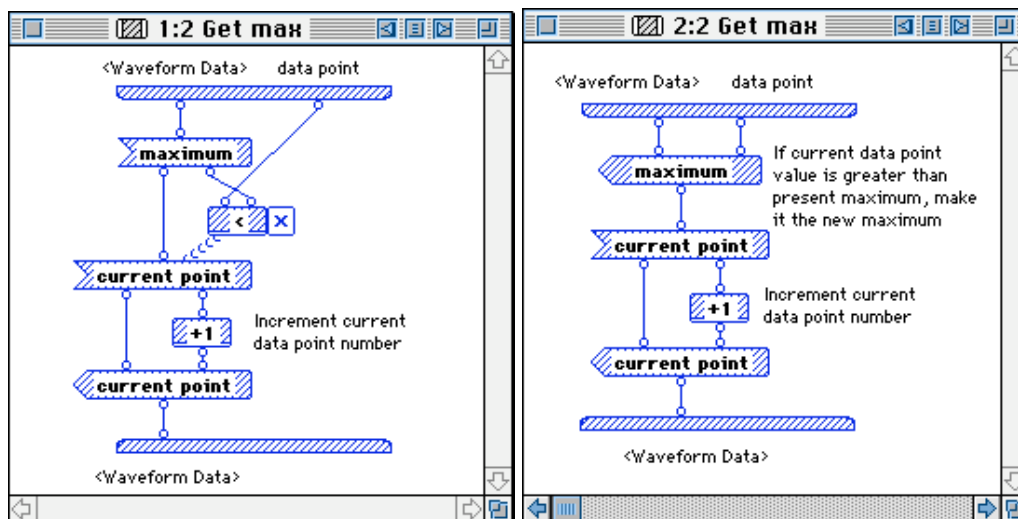


Figure 17.31: The Get max local method

The third class method of the Waveform Data class, Find Minimum, is essentially the same as Find Maximum except that the value of the minimum attribute is first set to the *highest* expected data value (see Figure 17.32).

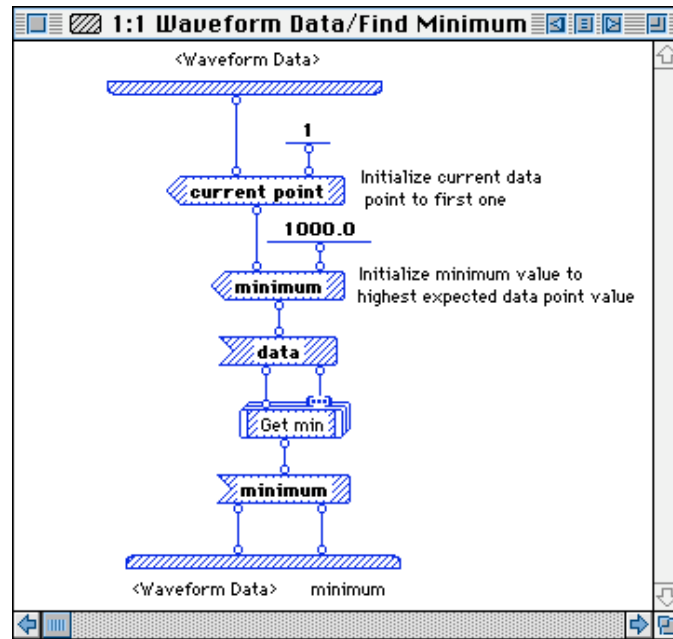


Figure 17.32: The Find Minimum method of the Waveform Data class

The final graph scaling method of Waveform Data is Find Range, shown in Figure 17.33, which calls Find Maximum and Find Minimum, then takes their difference as the peak-to-peak range of the data to be plotted. The range will be used to scale the data for the Waveform View.

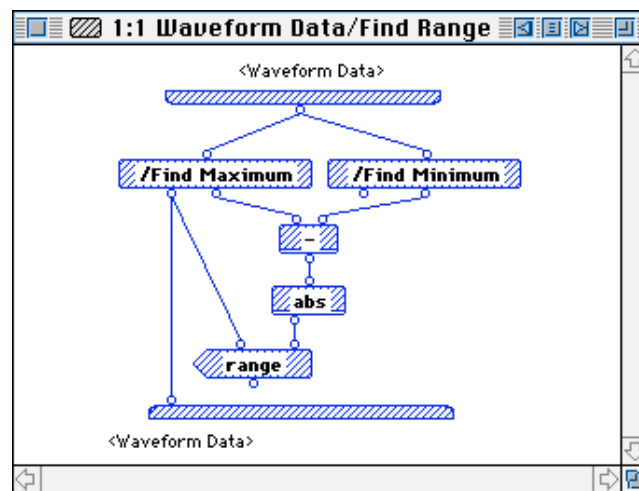


Figure 17.33: The Find Range method of the Waveform Data class

The **Get Value** method of the **Document Data** class is responsible for retrieving the contents of the document. In the **Document Data** class, this method simply returns a **NULL** value. This forces you to override the method in your **Document Data** subclass. In other words, **Document Data** is an abstract superclass which must be subclassed to be used in a program. We override the **Get Value** method in our **Waveform Data** subclass to just pass through the entire **Waveform Data** object as the contents of the document (see Figure 17.34).

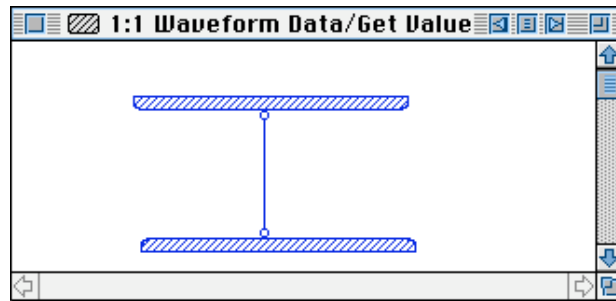


Figure 17.34: The Get Value method of the Waveform Data class

We must also override the **Set Value** method of the **Document Data** class so that we may alter the contents of the document. In the **Document Data** class, this method is empty. We override the **Set Value** method to get new values for the document data from another instance of **Waveform Data** that is input to the method (that is, a temporary object), then stuff the values of its attributes into those of the **Waveform Data** object that holds the contents of our document (see Figure 17.35).

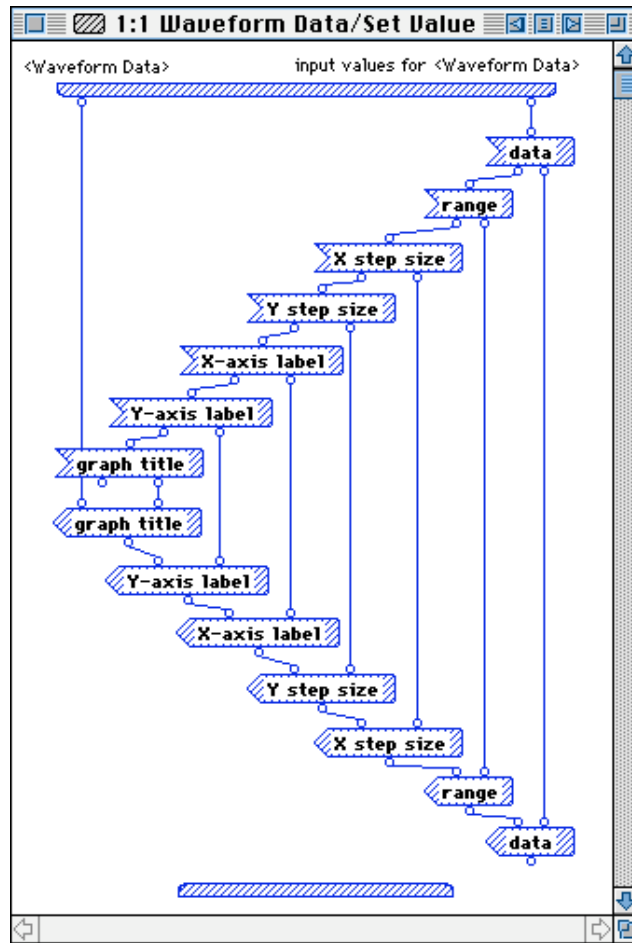


Figure 17.35: The Set Value method of the Waveform Data class

The **Waveform Window** class will need one method to help it draw the labels of the axes and the plot title. This method simplifies other methods that we'll write soon to set the labels of the plot by placing their common code for setting the string of a **Text** object into a single method. The **Set Label Text** method (see Figure 17.36) accepts an instance of the **Waveform Window**, and finds a **Window Item** (a **Text** object) whose name we supply as the second input to the method. The **Text** object's text is then reset to display a new string from the third input of **Set Label Text**.

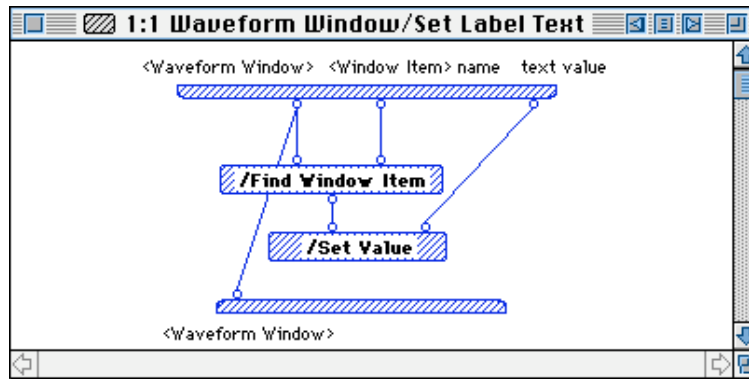


Figure 17.36: The Set Label Text method of the Waveform Window class

To help get and set data point values for the **data** attribute of the **Waveform Data** class, we create a new class named **List Items Mapping**, stored in a section of the same name. This subclass of the **Window Item Mapping** class may seem familiar to you as a class from the bar chart program in Chapter 11 of the Prograph Tutorial manual. Two methods of **Window Item Mapping** are overridden in this subclass so that *lists* of data may be retrieved from the document data and written to it. Remember that the **data** attribute of **Waveform Data** contains a *list* of data point values to be plotted. The **Set Attribute** method places the contents of a list into a list attribute, while the **Set Window Item** method gets the contents of a list attribute and places them into a **Scroll List** object in a window. We will not reproduce these methods here, but refer you to the Prograph Tutorial manual.

The **Waveform Document** class contains just three methods. The first is the **Draw** method that is called as the click behavior of the **Graph Data** button (Figure 17.37). It calls the **Get Data** method of the **Waveform Document** class to retrieve the contents of the document, then calls the **Waveform View** class' **Set Value** method to draw the graph. We'll discuss that method shortly.

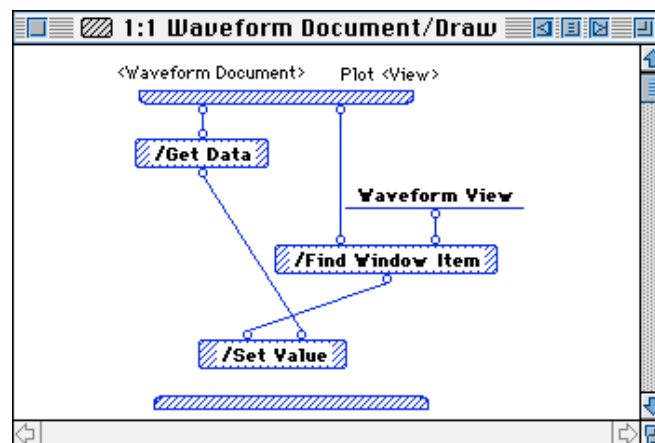


Figure 17.37: The Draw method of the Waveform Document class

The Add Data method (Figure 17.38) gets the contents of the Scroll List that displays the data point values for the graph and attaches a new data point to it, then calls Extract Data to place this new data list into the document data itself. The call to the Make Dirty method of the Document class marks the document as being in need of saving since its data has been changed.

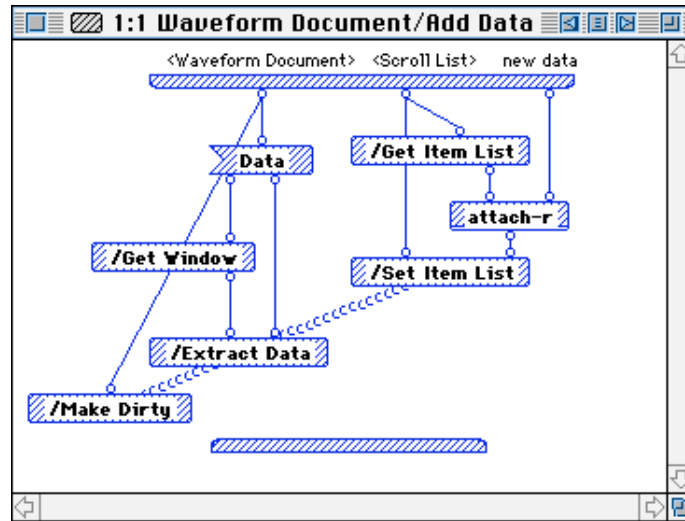


Figure 17.38: The Add Data method of the Waveform Document class

Remove Data performs the opposite action (see Figure 17.39). It gets the items selected by the user in the Scroll List (by calling Get Select List method of Scroll List to handle multiple selections in its list) and deletes those items from the list. It then calls Extract Data and Make Dirty as did the Add Data method.

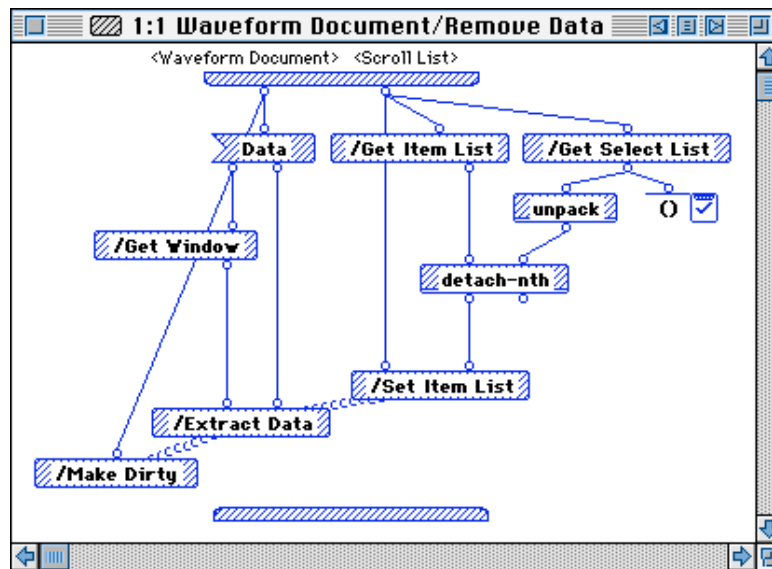


Figure 17.39: The RemoveData method of the Waveform Document class

We've finally arrived at the workhorse of the Data Plotter program -- the **Waveform View** class. We add seven new attributes to those inherited from the **View** class, shown in Figure 17.40. An instance of the **Pen** class is placed in the **Waveform View** class so that we can change aspects of drawing such as the width of a line being drawn. The next two attributes, **data point H offset** and **data scale factor**, determine how far to move along the x-axis and y-axis for the next data point in the plot. The **plot Y-maximum** and **plot Y-minimum** attributes determine the highest and lowest values that can be represented in the plot (rounded off to the nearest 100 units) and are used not just for scaling the plot, but also for setting the Y-axis numerical labels. **Plot X-maximum** sets the x-axis label for the highest x value (the lowest value is always set to zero in this program). The last attribute, **plot V offset**, moves the plot downward so that a zero value in the plot lines up with where the zero point of the y-axis should be.

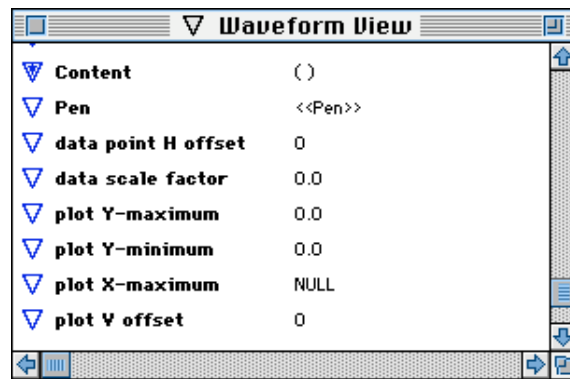


Figure 17.40: The attributes of the Waveform View class

The **Calculate Scale** method (Figure 17.41) determines the plotting scale for the data plot; that is, the correspondence between the units of the plot's x-axis and y-axis versus the horizontal and vertical extents of the Waveform View. It finds the vertical size of the **Frame** of the View (its visible extent rather than the area of the view that could be scrolled) and divides it by the **range** of the data rounded out to the nearest 100 data units.

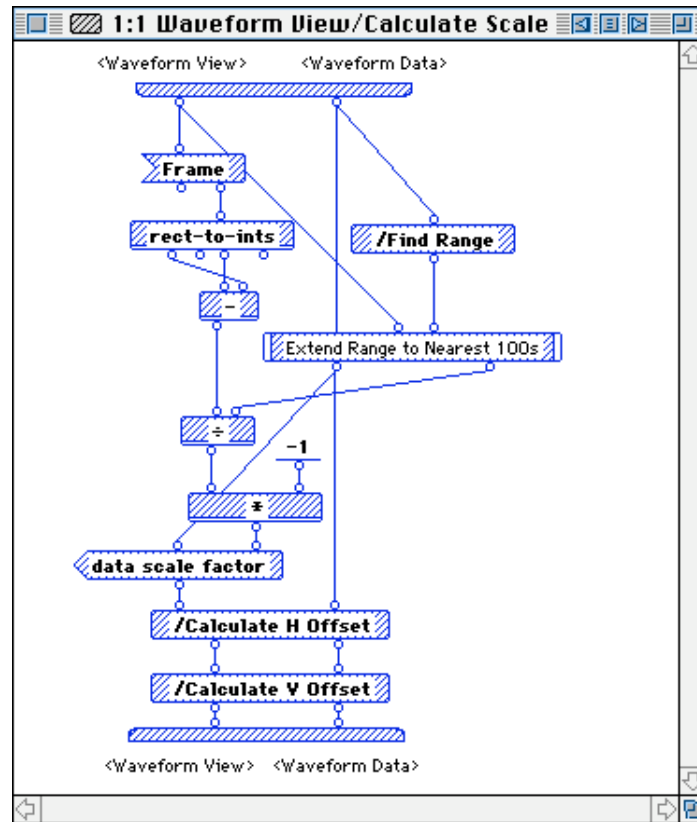


Figure 17.41: The Calculate Scale method of the Waveform View class

The Extend Range to Nearest 100s local method, shown in Figure 17.42, is responsible for making the plot “prettier”. It ensures that the upper and lower limits for the y-axis are whole multiples of 100 by rounding up the maximum data value and rounding down the minimum.

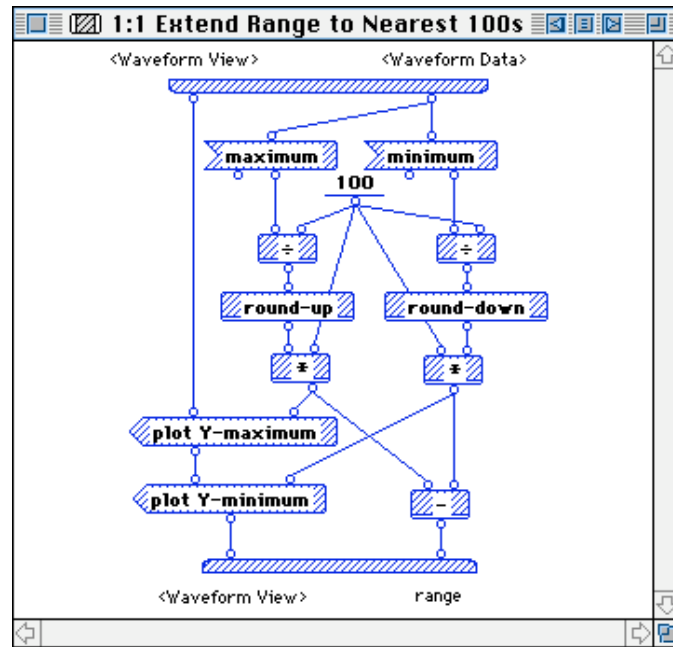


Figure 17.42: The Extend Range to Nearest 100s local method

Calculate H Offset determines the horizontal spacing of adjacent data points in the Waveform View (see Figure 17.43). It divides the horizontal extent of the Waveform View's frame by the number of data points.

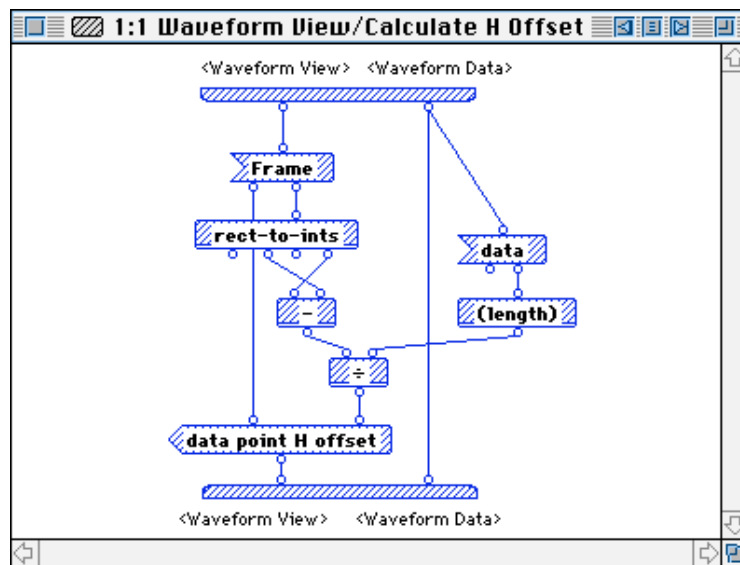


Figure 17.43: The Calculate H Offset method of the Waveform View class

Calculate V Offset calculates how far to displace the data plot downward in the Waveform View to get to the data plot's zero y-value (Figure 17.44). It divides the vertical extent of the Waveform View's frame by fraction of the data's range that is for positive values.

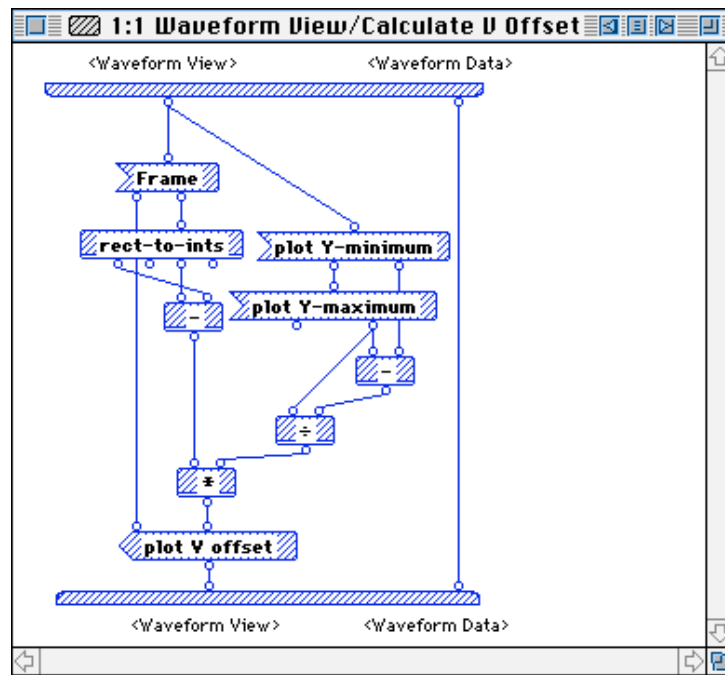


Figure 17.44: The Calculate V Offset method of the Waveform View class

Once we're ready to plot the data in the window, we must scale each plot data point's value to fit into the Waveform View. The **Scale Data To View** method, depicted in Figure 17.45, does just that. For each point in the plot, it multiplies the value of the data point by the **data scale factor**, and offsets its horizontal position in the Waveform View with the value of **data point H offset** and its vertical position with the **Offset Plot** method. **Scale Data To View** then moves the horizontal position of the point over by the **data point H offset**. The vertical and horizontal position of the current data point is then stored in point format and output from the method.

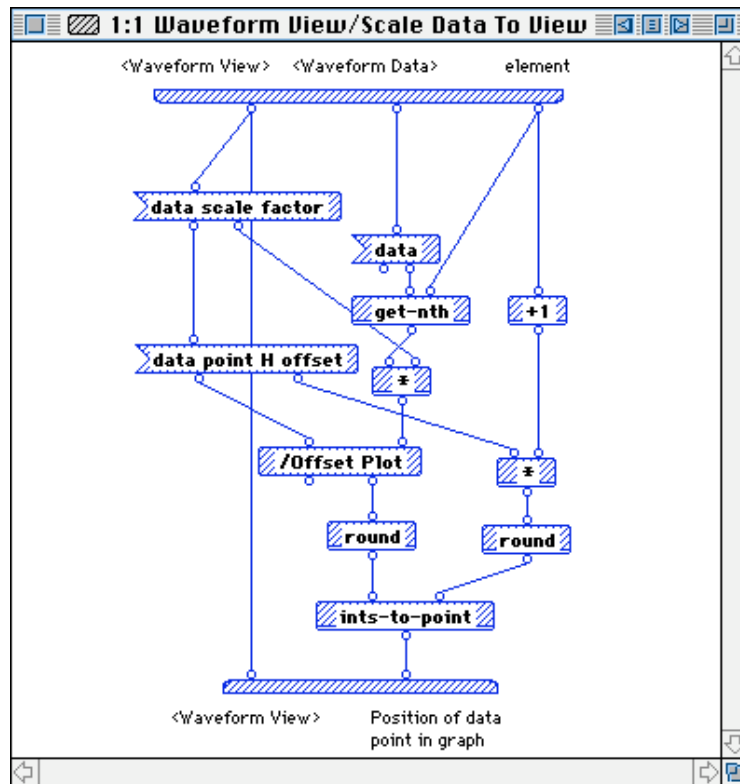


Figure 17.45: The Scale Data To View method of the Waveform View class

Offset Plot simply adds the plot V offset to the vertical position of the current data plot point (Figure 17.46).

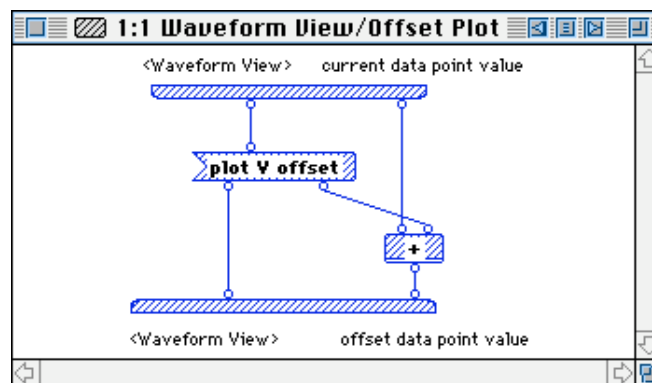


Figure 17.46: The Offset Plot method of the Waveform View class

What about drawing the plot? Well, this is what we've been waiting for! Let's start with the utility methods that will be called by the major methods that coordinate the drawing. These more focused methods will draw specific parts of the waveform graph display. The **Draw Waveform** method (see Figure 17.47) is the method that the Waveform View uses to plot the data in a graph. **Draw Waveform** first calculates the scale for the plot, then draws the lines that form the axes of the plot. Next, the **Pen** object

is used to change the thickness of the plotting line. The default pen size (line width) is one pixel. We set it to two pixels wide with the **Set Size** method of **Pen**. We then make the current drawing use the redefined pen with the **Pen's Use** class method. The waveform plot is then drawn via two local methods.

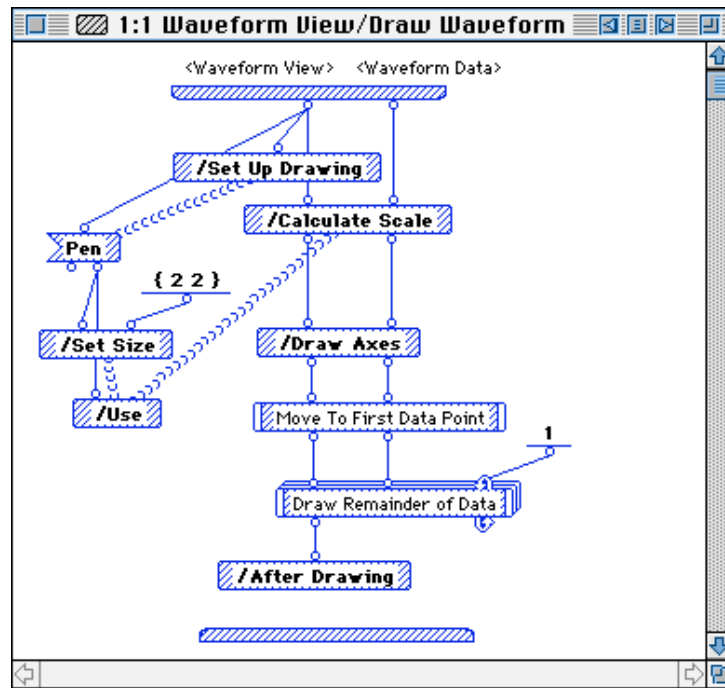


Figure 17.47: The Draw Waveform method of the Waveform View class

Before we continue with the drawing code, we should mention that the coordinate system for drawing in the Waveform View is from (0,0) at its top left corner to (width,height) at its bottom right corner. Unfortunately, the **Frame** attribute of the **Waveform View** class has its origin not at (0,0) relative to the *View itself*, but at the position of the top left corner of the Waveform View *relative to the window's coordinates*. So if the Waveform View's top corner is 20 pixels from the top of the window and 10 pixels from its left, **Frame's** top left corner is at (20,10), not (0,0) as we'd expect. Therefore, in some of the methods to follow, we must subtract out the frame's left position (for horizontal position calculations) or top position (for vertical position calculations) when drawing in the Waveform View.

The first of the two local methods of **Draw Waveform** is **Move To First Data Point**, shown in Figure 17.48. It scales the first data point, then moves the drawing “pen” (the drawing location) to the first point's location.

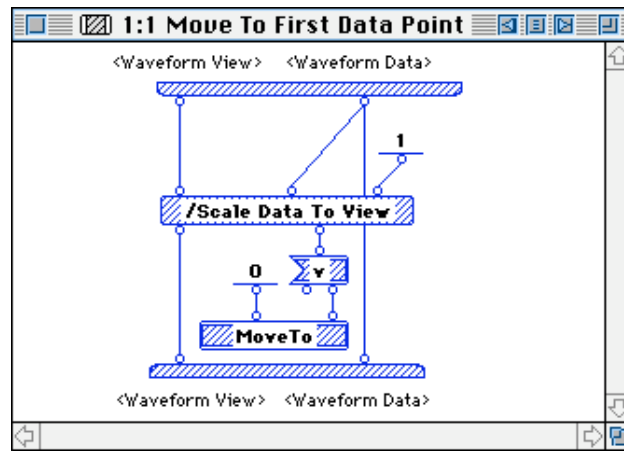


Figure 17.48: The Move To First Data Point local method

The Draw Remainder of Data local method, as its name suggests, draws the points of the data plot (see Figure 17.49). This method is looped, once for each data point, and simply draws a line from each data point to the next after scaling the point for the Waveform View.

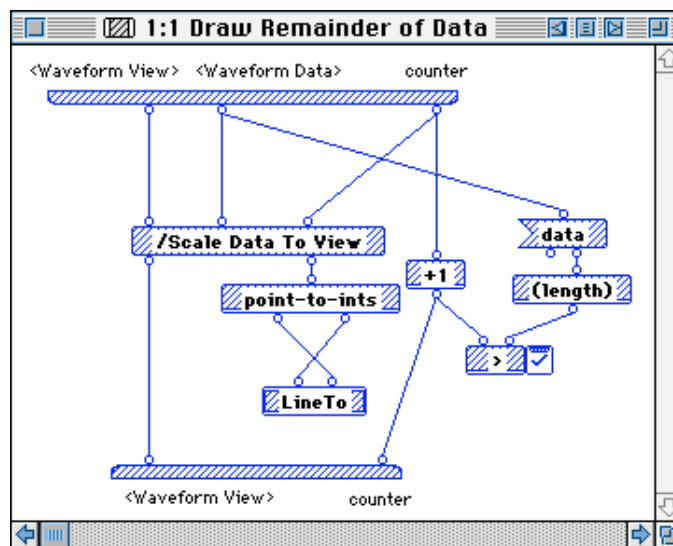


Figure 17.49: The Draw Remainder of Data local method

To make the data plot look like a graph, we draw the axes of the plot into the waveform View with the Draw Axes method (Figure 17.50). This method calls two local methods, shown in Figures 17.51-17.52, which draw lines along the left and bottom edges of the Waveform View.

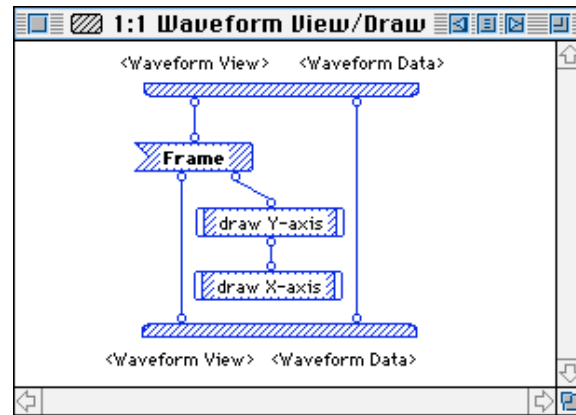


Figure 17.50: The Draw Axes method of the Waveform View class

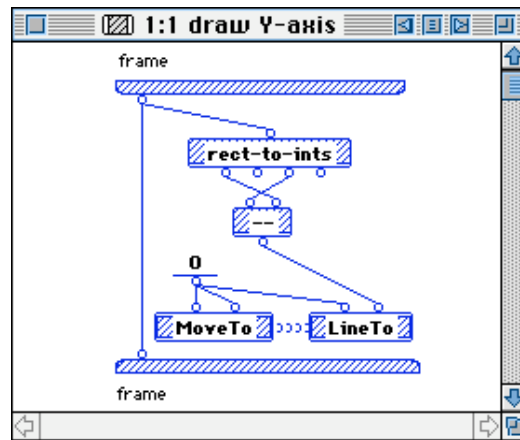


Figure 17.51: The draw Y-axis local method

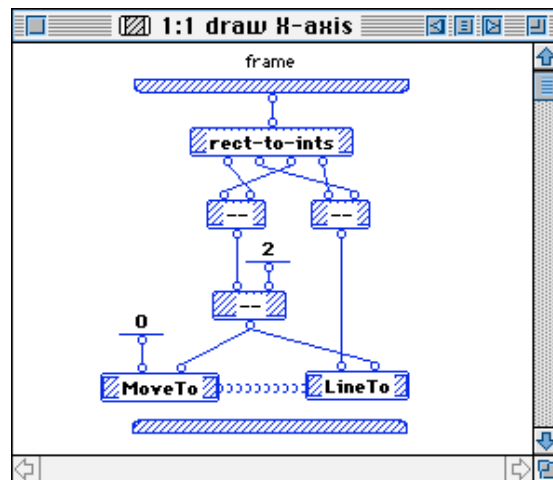


Figure 17.52: The draw X-axis local method

The **Draw Labels** method of the **Waveform View** class sets the text of the labels for the data plot. It calls two local methods (see Figure 17.53). The first uses attributes from the **Waveform Data** class to set the labels of the x- and y-axes as well as the title of the plot. The second reads attributes of the **Waveform View** class to fill in the text showing the minimum and maximum values for the x- and y-axes.

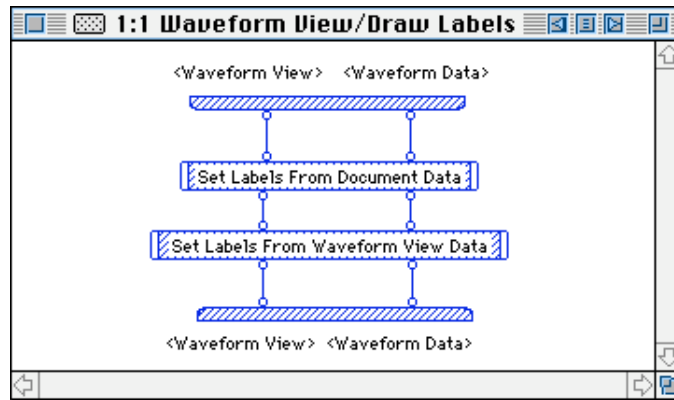


Figure 17.53: The Draw Labels method of the Waveform View class

The **Set Labels From Document Data** local method (Figure 17.54) gets the values of the **data type**, **y-axis units** and **x-axis units** attributes of **Waveform Data** then stuffs them into the appropriate **Text** objects in the **Waveform Window** using the **Set Label Text** method of the **Waveform Window** class.

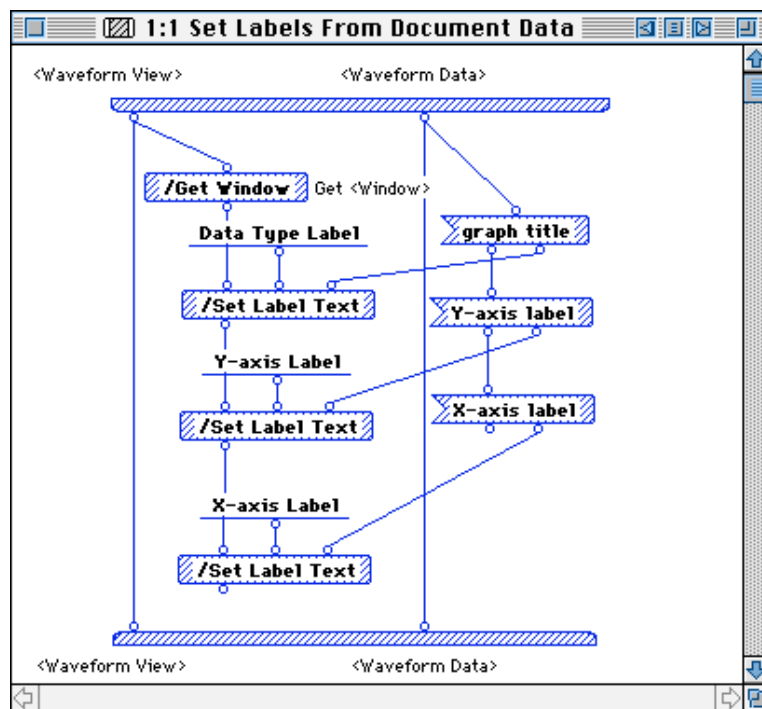


Figure 17.54: The Set Labels From Document Data local method

Set Labels From Waveform View Data (Figure 17.55) gets the values of the plot X-maximum and plot Y-maximum attributes of Waveform View, multiplies them by X step size and Y step size, respectively, then sets the values of the Text objects in the Waveform Window that display the minimum and maximum x-axis and y-axis values using Set Label Text again.

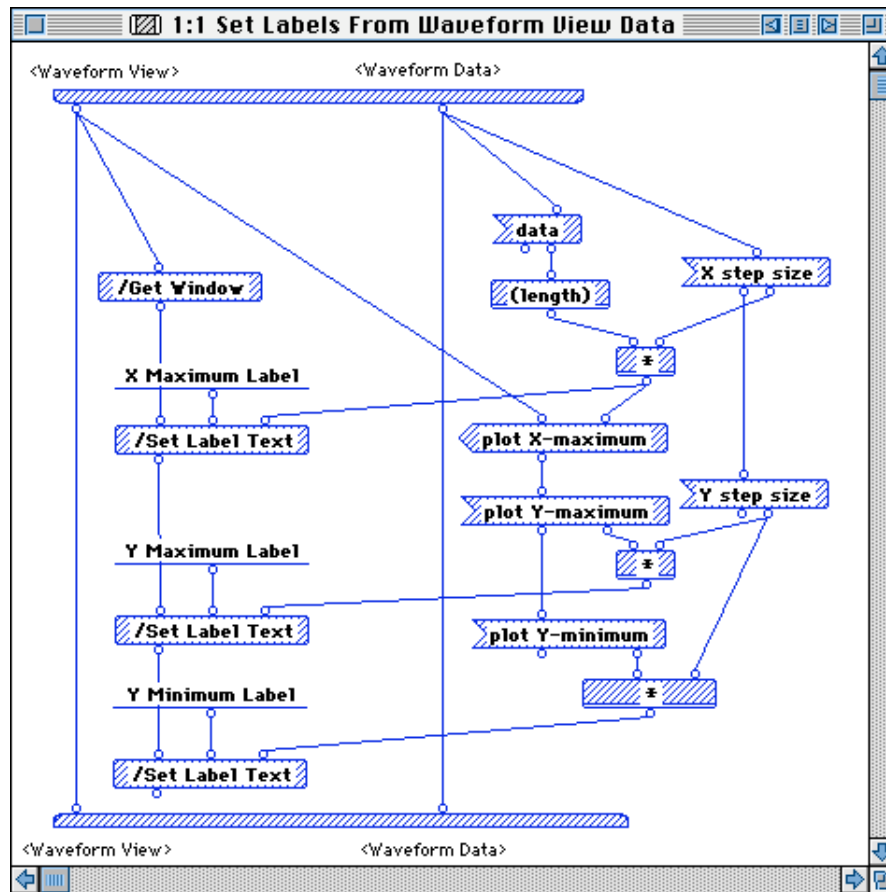


Figure 17.55: The Set Labels From Waveform View Data local method

Now we are ready to look at the methods that handle the drawing. Remember that the Draw method of Waveform Document (Figure 17.37) called a class method of the Waveform View class called Set Value. This method, shown in Figure 17.56, overrides the Set Value method of the View parent class so that it redraws the labels of the waveform graph, then sends a request to refresh the contents of the View window item, which will be done by calling the Waveform View's Draw method (shown later in Figure 17.59).

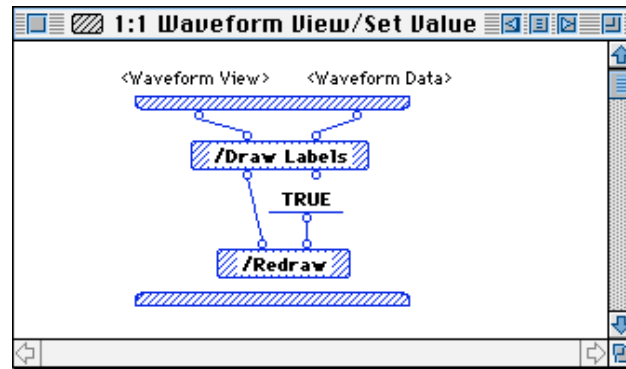


Figure 17.56: The Set Value method of the Waveform View class

The Graph Data method, depicted in Figure 17.57, is the method that coordinates the chore of drawing into the Waveform View. It first checks that data actually arrives into the method by comparing the input value of the data to NULL. If no data arrives, the method exits. One thing to note about this method is that sometimes when it's called, a **Waveform Data** object arrives as its second input, but at other times a **Waveform Document** object arrives. How does the method handle both of these situations? It calls a local method named **Ensure data** that outputs a **Waveform Data** object regardless of the method's input. The data is then passed on to the **Calculate Scale** method, then the **Draw Waveform** method, which do the actual drawing of the waveform graph.

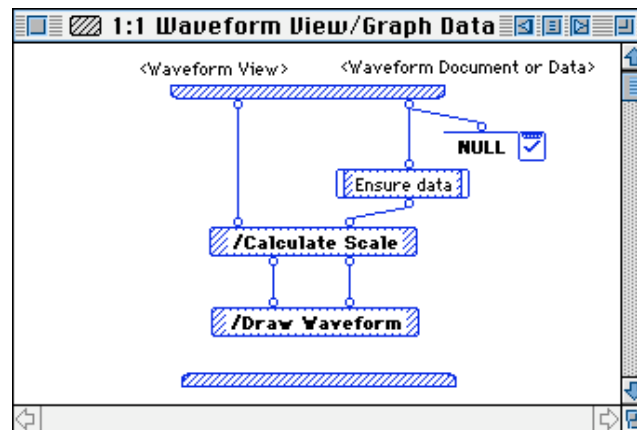


Figure 17.57: The Graph Data method of the Waveform View class

The **Ensure data** local method (see Figure 17.58) checks if its input data is a **Waveform Data** object by calling the **type** primitive, which returns the data type of any variable or constant. If the data is indeed a **Waveform Data** object, it is passed through in case 1. If it's a **Waveform Document** object instead, the **Get Data** method is called in case 2 to extract its **Waveform Data** object.

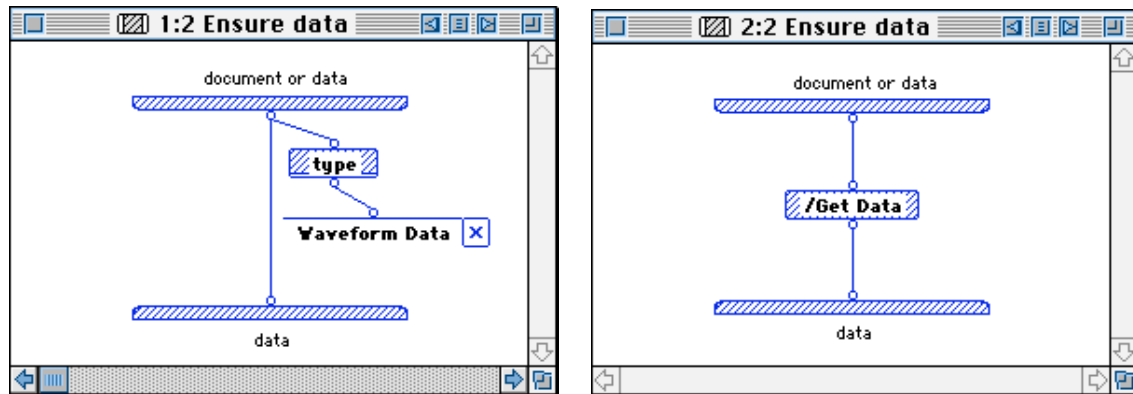


Figure 17.58: The Ensure data local method

Only two methods remain in the **Waveform View** class. These methods override class methods of the **View** class that handle requests to draw and print the contents of the Waveform View. They are named, not surprisingly, **Draw** and **Print**. The **Draw** method (Figure 17.59) calls the superclass' **Draw** method using the bounds of the area into which it must draw as its second input, then gets the window of the Waveform View and the owner of that window (the **Waveform Document**). Finally, it calls the **Graph Data** method to draw the graph into the Waveform View. The **Print** method, shown in Figure 17.60, performs nearly the same actions except that it first calls the superclass' **Print** method. This method is called indirectly by the **Print Layout** class when we select the **Print** menu item.

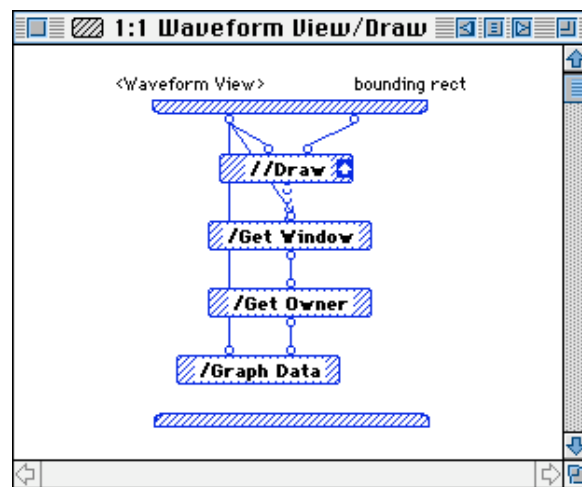


Figure 17.59: The Draw method of the Waveform View class

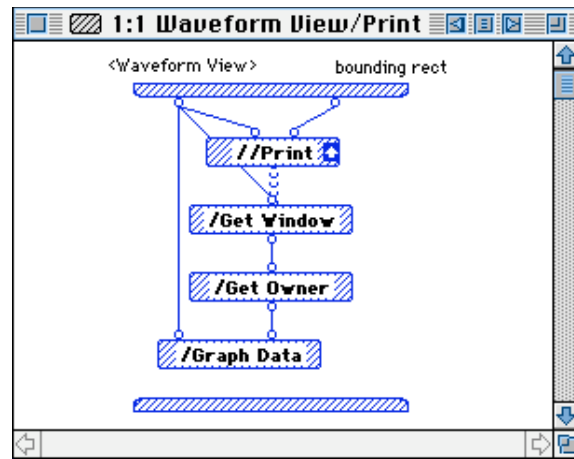


Figure 17.60: The Print method of the Waveform View class

This completes the Data Plotter program! As you look through its code, you can see that the actual amount of code that does the drawing is still pretty small for a graphics program. By using existing code in the ABCs, we've barely had to program at all. This is not an insignificant fact -- think about how many lines of code you'd have to write just to set up a document and a drawing of this kind in a textual language like C or C++.

This last program of the book was presented to hit you on the head with the one lesson we hope you'd learn about Prograph. Even the most complicated of programming tasks become easier and faster to program in Prograph. The combination of integrated visual tools and a visual language lets you solve any programming problem with very little programming man-power. It's about time that programming became not a torture, but a pleasure. Prograph brings back the *fun* to programming!

Exercise 17.1:

Create a second subclass of the **View** class named **Bar Graph View**. Using code from chapter 11 of the Prograph Tutorial manual and from the code from the **Waveform View** of this chapter, write class methods for the **Bar Graph View** class that will plot the waveform data as a bar graph. Create a second subview in the Plot View of the Waveform Window (a **Bar Graph View** in this case) that is the same size as and covers up the **Waveform View**. Make a Radio Button set for the Waveform Window that will let the user select which graph type to plot by setting the **Visible?** attribute of each **View** subclass.

Exercise 17.2:

There are many more user interface elements offered by Prograph that we have not demonstrated since we wanted to keep our discussion focused on realistic complete programming examples. These other elements are made as easy to use as push buttons and text-editing boxes by the ABCs and ABEs. In this final exercise, construct a Catalog program to keep track of a list of hardware products stored in a document. You should be able to display the individual hardware items in the catalog, add new products and remove outdated products, and print out information on a product. Two windows should be presented to the user: In the main Inventory window, which opens when the program starts, you should place a *scrolling list* of product names and *push buttons* for adding or deleting items from the list, as well as a Quit button. When an item in the scrolling product list is double-clicked, a second Product window should open for displaying the products. This window should contain *text-editing boxes* for the name of the product, its size and price, a *pop-up menu* for the type of product (electrical, paint, lumber, connectors, etc.), a *check box* for whether or not the item is in stock and a Close push-button to return you to the main window. Feel free to experiment and add other GUI elements to the window, such as *radio button sets*. Have fun exploring the ABCs!