Glyphic Codeworks™ Procedures Manual

Monday, November 15, 1993

Draft Notice

The following document is in draft form. It is not complete, but does present most of the operations available in the environment. Features which are likely to change and known problems are printed in italics.

This document is intended to be a companion to the demonstration version of Glyphic Codeworks. Along with the tutorial and language reference documents, it should be enough to allow people to use the system,

— Bruce Schwartz & Mark Lentczner
Glyphic Technology
http://www.glyphic.com/

Glyphic Technology

Chapter 1

Base Procedures

System Organization

You create a Glyphic Codeworks application by creating classes, instances of classes, and folders.

Classes are prototypes of useful data. For example, a class 'wine-case' might be used to record information about a case of wine stored in a cellar. Or, a class 'auto' might be used to contain information pertaining to a used car for sale. A class describes the look and operation of the data, whether the computer has one or one hundred pieces of that kind of data. You work with the class whenever you want to change how the data operates or looks.

Instances of classes, commonly called 'objects', are particular pieces of data. Continuing the example above, a 'wine-case' object would be used to record 12 bottles of estate red, or 'car' object for a '57 Chevy in mint condition. Each object represents one entry of data. Each object has a class, from which it inherits the way it looks and operates. An object is often referred to as an instance of its class. The class of an object is called its parent.

Folders are groups of objects or classes. Each folder can contain any number of objects. Folders can hold objects of only one type, or can hold objects of different types. A folder is like a section in the Notebook.

Classes and folders are really just objects themselves. In this manual, whenever the term object is used, it applies equally well to classes and folders unless otherwise stated.

Codeworks stationary documents start with six folders. They are:

- **user classes** contains all new classes that you create
- **miscellaneous** is an empty folder that can be used for new objects
- **trash** contains deleted objects before they are purged
- **system objects** contains several useful objects that are already defined in the system
- **system classes** contains classes already defined in the system
- **system drawing** contains classes and objects useful in drawing and user interface

The last three folders are fixed and can't be changed.

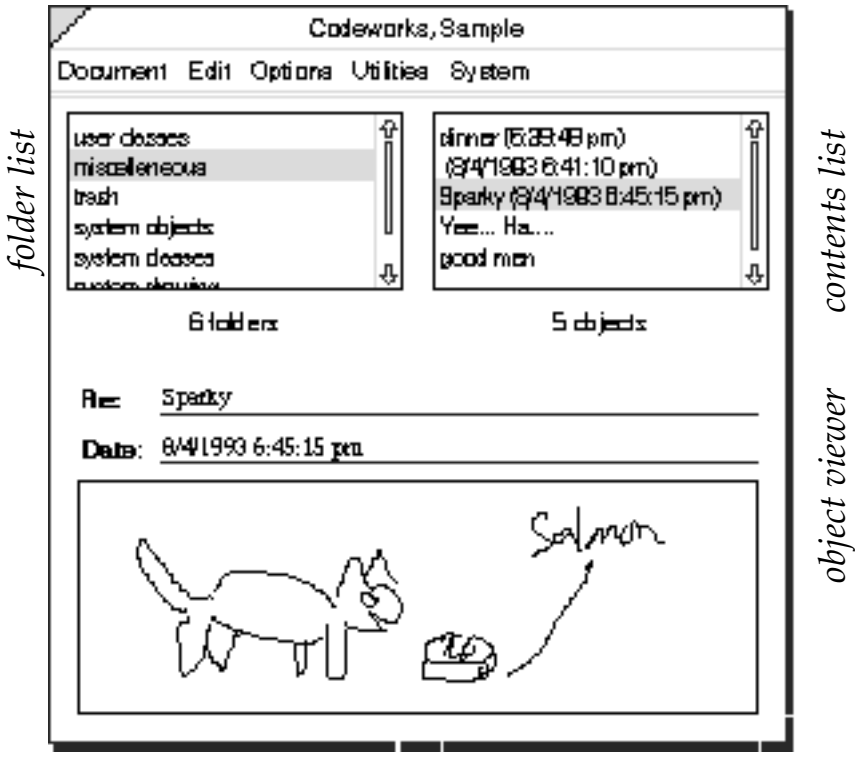
Browsing the System

The folders, classes and object in a Codeworks document can be explored with the folder browser. If you create a new, blank, Codeworks document the folder browser will be displayed in the main window of the document. If open some of the sample applications a different view will in the main window. In these cases, the folder browser can be opened floating.

To open the folder browser:

1. **Tap on 'Folders…' in the Utilities menu.** The folder browser opens in its own floating window.

The folder browser has three areas. At the top left is *the folder list* which lists all folders in the system. Selecting a folder in the folder list causes the top right section, *the contents list*, to display the contents of that folder. In general, selecting an object from the contents list displays the object in the bottom half of the folder browser, *the object viewer*. If the object is a user class, then selecting an object from the contents list will open the class up into its own window, ready for editing.



Opening and Displaying Objects

Objects are usually displayed in the object viewer. Objects can also be opened into their own floating windows. To open an object into its own window double tap on an object and then choose 'Open' from the menu that appears.

An object may be displayed in one of several different formats. The system will choose a format for an object when ever it needs to display it. There are two major types of formats:

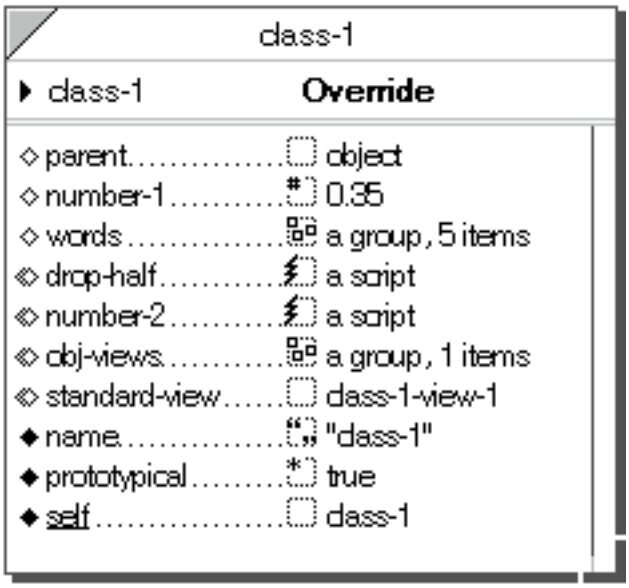
- **browsers** are a list of all the internal properties, values and scripts of an object. These formats are primarily used when writing scripts.
- **views** are graphical layouts of an object's properties, often like a paper form, and sometimes like a picture. These formats are what the end-user of your application will see.

You can explicitly open an object in a particular format with the 'Open…' command. It displays a list of formats (including one or more browser types). Choosing one of these opens the object in that format.

Working with Browsers

When looking at a object in a browser you can see all the properties of the object and their values. Each property also displays its type with an icon, and each value gives an indication of what kind of object it is with an icon. Each entry in the browser list is divided into two halves, each with its own menus: The left side menus operate on properties, the right side menus operate on the values.

In addition, the last item in the list is **self** which is an entry for the object the list is displaying. This isn't really a property of the object. It is just added to the list for convenience (for example, you can move the object by Pressing on it there).



Property Type

- instance
- class
- private

Value Type

- object
- number
- string
- group, array
- script
- nil, ???, true, false
- project
- project, changed
- external
- call stack entry
- script text

The Browser Stack

Opening a value from a browser into a browser view (via either the 'Open…' command, or 'Open' command if the object has no standard view), will *not* display the object in a new floating window, but will instead reuse the browser window.

You can think of a browser as a stack of pages, each showing one object. The pop-up list at the top of the browser window keeps track of

the path of objects that brought the browser to currently viewed object. You can display any of the objects in this list by tapping on it.

Tip: Flick right and flick left on the pop-up menu quickly move up and down through the stack of objects.

Adding & Editing Properties

When you add a property or edit a property (Caret or Check on the left hand side) a modal note appears. In the note you can edit the name of the property, select the scope type, and choose to make the property a script.

If you make the property a script, then a script editor appears. In this case, the property is not added or changed until you enter and save the script. As a shortcut, if you choose to make the property a script, you need not select the 'Class' property type. All scripts will be made 'Class' by default, even if you don't select it.

Floating a Browser

You can tear the current object off a browser stack and float it in its own window. This will create a second browser, with its own path of objects, starting at the torn off object.

To float the displayed object in a browser:

- **Up-caret in the title bar of the browser.**

Expression Editors

Some objects (including scripts, numbers and strings) open up into editors (such as pen input pads, or small text editors) in which you edit the value. In these cases you are not editing the object, but rather the property that contains the object.

For example, if you tap on the number 7 in some view, a small window will appear in which you can edit the number to a different value, such as 42. You are not changing the number 7 into the number 42, rather you are replacing the 7 that was in that property with the number 72.

If you have changed the text of an editor and try to change the object the browser is showing, or close the browser, you will be asked if you want to save or discard the changes first. When you change an instance variable to a script, it automatically becomes a class variable when you save it.

Group Browsers

Group browsers are a kind of property browser used for groups and arrays. Instead of a property names and scopes, they display the numeric index of each value on the left side. Since groups don't have properties, and can't have scripts as values, only the right side menus are available in group browsers.

Creating New Objects

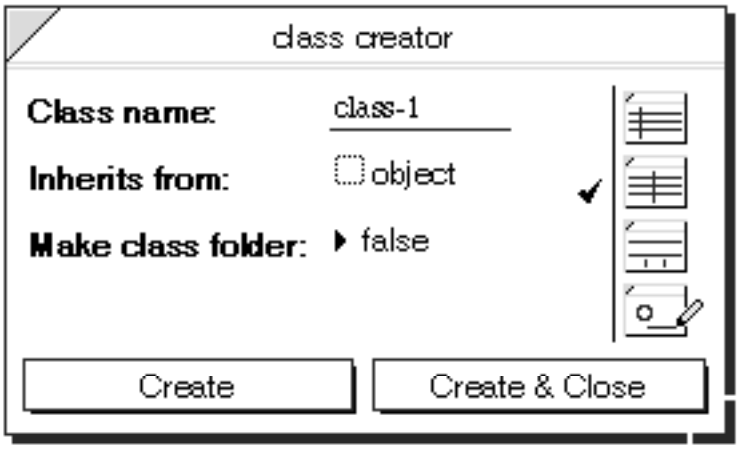
All of these procedures are performed in the folder browser. You may need to open the folder browser if it is not displayed in the main window. To do this, tap on 'Folders…' in the Utilities menu.

Creating a New Class

New classes are created in the user classes folder. Every user class must have a name and view. Before creating a class, decide what you want to call it and what it should look like.

To create a class:

1. **Tap on the user classes folder in the folder list.**
2. **Up-Caret in the contents list.** The class creator window appears.
3. **Fill out the class creator window options.** The required options are: the name of the new class and the initial view (from the iconic list). Other options are the object the new class inherits from and a class folder (which is a folder expressly for instances of the new class).
4. **Tap Create & Close.** The new class appears in the user classes folder and opens into its own window, ready for editing.



Creating a New Object.

1. **Tap a folder in the folder list.** This is the folder where the new object will appear. The current contents of the folder will appear in the top right section of the folder browser. The miscellaneous folder is a good place for new user objects.
2. **Up-Caret in the contents list.** If the folder is specifically for a particular kind of object (user classes for example) then the object is created; skip the next step. Otherwise, a menu of classes appears.
3. **Tap on the kind of object you want to create from the list.** The object is created and added to the list.
4. **Tap on the newly added object in the contents list.** The new object is displayed in the object viewer and can be edited there.

Creating a New Folder

1. **Up-caret in the folder list.** A note appears asking you for the name of the new folder.
2. **Write in the name of the new folder.**
3. **Tap OK.** The new folder appears in the folder list.

Deleting Objects

A Codeworks document generally retains all objects that you create until you explicitly delete them. Individual objects can be deleted or whole folders can be deleted at a time. When you delete objects from most folders they are actually just moved to the trash folder. You can move them back to some other folder if you really didn't want to delete them. Objects are only completely deleted when you delete them from the trash folder as well.

Note: objects can appear in more than one folder at a time. Objects are only deleted when they are removed from all folders, including the trash folder. Furthermore, an object won't be deleted if any other object in the system still refers to it. For example, you can't really delete a user class if you still have instances of that class in folders. The instances of the class refer to the class object (it is their parent) and will keep the class from being deleted.

Deleting an Object

To delete an object from a folder:

1. **Tap on the folder in the folder list.**
2. **Cross on the object in the contents list.** In general the object is moved to the trash folder.

Deleting all Objects in a Folder

To delete all objects from a folder:

1. **Double tap on the folder in the folder list.** A menu appears.
2. **Tap 'Delete All' from the menu.** A warning note appears.
3. **Tap 'Yes' in the note.** You can cancel the operation at this point by tapping 'No'. In general, all the objects are moved to the trash.

Clearing the Trash

You can use either of the two procedures above for clearing out the trash folder. Items deleted from the trash are removed from the system if there are no longer any references to them. If any folder or ob-

ject still refers to an item, then it is only removed from the trash.

Exploring System Objects & Classes

Much of the functionality available in Glyphic Codeworks is in the objects and classes that come standard with the system. These are commonly used in programming and building your own objects. You can explore how existing objects are programmed to understand these objects better. The scripts for most of the system are available.

Most of the more interesting objects are in the system folders visible in the folders list of the folder browser. These objects are organized into useful categories. Start exploring these objects first. *[The organization is still under construction.]*

The full collection of all system objects can be found via the projects browser. To open it tap on 'Projects…' in the Utilities menu. A browser on the projects list will appear.

The projects are:

- **browser-project** implements much of the functionality of the environment and provides several useful messages for performing script cross references
- **doc-project** the open document
- **folder-project** implements folders and the folder browser
- **layout-project** see view-project
- **root-project** implements the basic objects available in the system, including **object**, **group**, and **number**
- **penpoint-project** provides an interface to PenPoint's own objects
- **standard-project** see view-project
- **user-prim-project** implements a file access object that uses the example user-primitive C program
- **view-project** along with layout-project and view-project, implements the view system

Moving & Copying Objects

Move and Copy work a little differently with objects than they do in the rest of PenPoint. Since an object may be referenced from several objects at once, an object can't really be moved from one object to another. Therefore, in Glyphic Codeworks, when you move an object from one place to another, the new place refers to the object, and old place continues to refer to it as well.

Moving an Object in a View

Press on the object until the move marquee (single dashed lined) appears. Drag the marquee to the destination. The destination now refers to the object. You can move objects from several places: from the right side of property and group browsers, and from form views in Author mode.

Remember that some objects open into editors that change the property holding the object, not the object itself. For example, if you move the string "aardvark" from one object to another, then edit it to read "zebra", the original object will still refer to the string "aardvark". This is because editing a string doesn't change the string itself, it replaces the reference to the string with a reference to a new string.

Moving a Displayed Object

From a property browser:

1. Press on the 'self' entry at the end of the property list.

From a group browser:

1. Press on the entry after the end of the list.

From a form view:

1a. Double tap on the title line to get the view menu.

1b. Tap on Move.

Then:

2. Drag the marquee to the destination. The destination now refers to the object.

Copying an Object

The procedure is just like the procedure for moving, only use tap-press instead of press. In the case of copying an object displayed in a form view, choose Copy instead of Move from the view menu.

Some objects treat copying specially. Copying fixed objects (like numbers, true, false, and symbols) is just like moving them, since there can only be one of each of them in the system.

Moving and Copying Scripts

Moving and copying scripts will always immediately open up a script editor. In order to complete the operation, tap the Save button at the top of the script editor, editing the script if needed. Moving is the same as making a copy: a script can only be referred to from one object at a time.

Renaming an Object

All objects have a name that is used when the object is displayed. The name is normally supplied by the system, or you can choose to rename it.

Renaming from a Property Browser

Circle on the object to be renamed. Be sure to make the circle on the right half of the list over the object to rename, otherwise you will be renaming the property. Edit the text in the edit pad that appears.

Renaming an Open Object

When the object is open in a property browser or a form view:

1. **Circle on the title line.**
2. **Edit the text in the edit pad that appears.**

Reverting to the System Supplied Name

Use the procedures above to rename the object, except that when the edit pad appears, tap Clear and tap OK. This will remove any name given to the object and let the system supply an appropriate name.

Workspaces

Workspaces are useful small objects for working with objects. Workspaces can be used to:

- **try small parts of a script**
- **send messages to objects**

Creating a Workspace

In the Utilities menu of the document main window, tap on New Workspace…. A new workspace will open in its own window. The workspace will be added to the miscellaneous folder, so if you close it, you can find it again there.

The upper portion of the workspace window is a property list view of the workspace object. The lower portion of the window is a script editor where you can write expressions in Glyphic Script and run them. For more information on how to use these kinds of views, *see Property Lists and Script Editors in chapter 4, A Catalog of Views.*

Running a Script

1. **Write some text in the lower, script editor, portion of the workspace.**
2. **Select the text to be executed.**
3. **Tap on the Save button at the top of the workspace window.**

When you tap Save the selected script is executed as a property of the workspace object. This means that the script you write can access the properties of the workspace by name. Several properties are predefined:

- **a, b, and c** are properties are for your use. You can use them as temporary holding places for the objects you are working with.

- **result** holds the result of running the selected script. If you want to save a result, be sure to drag it to another place, such the **a**, **b**, or **c** properties.

- **code-text** holds a copy of the workspace's text. It is updated every time you save the workspace. Leave this property alone, if you change it in any way, the workspace may not remember the text the next time it is opened.

Note that scripts in a workspace behave a little differently than scripts elsewhere in the system. First, only the selected portion of the text is run in a workspace. Second, even if you don't use the word 'return' at the end of the selected portion of the text, the result of the last expression is returned anyway.

If running a script did not produce a result (or the result was undefined), then the **result** property gets set to the string 'No Result' to let you know.

Standard Menus

Double tapping on almost any area of the screen in Glyphic Codeworks will open a menu of applicable commands. Different objects have different commands, and different parts of an object in some views have different commands.

The list of commands is often broken up into two or more sections. These sections are listed in bold. Generally, only one section's commands are listed at a time. Tap on a bold section name to show the commands in that section.

This section lists all the commands that exist and a brief description of what they do. Generally, longer descriptions can be found elsewhere in the manual. You can find out if a given command is available for an object by Double tapping on it to display the command menu.

Some commands have gesture equivalents listed in the menu. Performing that gesture on the object is the same as Tapping on the command from the menu.

Object Commands

These commands operate on an object. They are generally available from title bars, browsers and object views.

- **Open** opens the object in its standard view.
- **Open...** displays a pop-up choice of all ways the object can be viewed (including a browser). Picking one opens the object in that way.
- **Rename** displays an input pad to rename the object.
- **Move** starts a move operation. Generally this moves a reference to the object, leaving the object where it started.
- **Copy** starts a copy operation.
- **New** starts a copy operation, but the copy is a new instance of the class. For classes, this menu replaces Copy.
- **Options** displays a pop-up menu of option sheets to choose from. Tapping on one opens that option sheet for the object.

- **Clear** sets the value of a property to ???. This is only available in browsers and object-views.
- **Toggle** replaces true with false, and false with true. This is only available for the values true and false.
- **Replace** lets you choose from a pop up list of objects to replace a value of ???. This is only available for the value ???.

Object Development Commands

These commands are generally available in title bars. However, these are only available in the development environment.

- **Open Browser** opens the object in a browser.
- **Edit Menus** opens a menu editor that lets you add menus for this object. For some objects, a pop-up list first appears displaying the different menu lists to edit. Tapping one opens and editor on that list for the object.
- **Override...** brings up the same menu as the Override menu in browsers. Choosing a message brings up an editor for writing that script.
- **Dump** displays debugging information for the implementors in the system log. *[This will not be present in the final product.]*

Browser property commands

These commands appear in browsers. They apply to properties and are available by double tapping on the property side of the items in the property browser. Although their names are the same in many cases to the commands above, they can be distinguished because they are all in the **Property** section of the menu.

- **Rename** displays an input pad to rename the property.
- **Edit** displays a note where you can change the property's scope and name.
- **Copy** or **New** starts a copy operation, as above.
- **Move** starts a move operation.
- **Insert** displays a note for adding a new property.
- **Delete** removes the property.
- **Implementors** finds all objects that implement the same property. This is also available from the title bar of scripts.

- **References** finds all scripts that reference the property. This operation may take awhile. This is also available from the title bar of scripts.

View Title Bar Commands

These commands are available from the title bar of a view of an object. They operate on the object as a whole. In either mode, most of the object commands listed above are also available.

- **Toggle Mode** Switches between user and author modes.
- **Layout** forces of he window. Available in user mode only.
- **Make Main Win** causes the view to take over the main window of the document. (The current main window floats.)
- **Open View Object** opens the object responsible for the view. This is only for view system programmers.
- **Open Display Object** opens the object responsible for this display of the view. This is only for view system programmers.
- **Edit Draw Script...** open the script that does the drawing of the view. Available only in drawable views.

Group Commands

These commands are available, where appropriate, in list views, browsers and folders. In addition, most of the object commands are also available for items in these groups.

- **Insert** adds an item to the group.
- **Delete** removes an item from the group.
- **Select** makes the item the selection of the group.
- **Delete All** removes all items from a folder. It verifies the operation before proceeding.
- **Create** adds an item to a folder. If there isn't a create class defined for the folder (see the folder's options), then a list of all user classes is displayed to choose from.

Misc View Commands

These commands operate on field inside a view when a view is in author mode. Many of these commands are available in only some views. Most of the object commands listed above are also available.

- **Rename Control** displays a note where you can rename the displayed name of the control (check box or button) separate from its property name.
- **Delete** deletes both the view and its property.
- **Delete View Only** deletes just the view.
- **View Options** displays a pop-up list of options for the view. Tapping on one opens that options sheet for the view.
- **Insert View** displays a pop-up menu of the items in the Palette. Tapping on an item, inserts a copy of that view into the aligner. Available in aligners.
- **Edit Choices** opens the group of available choices for a choice view, pop-up choice view, or multiple-choice view.
- **Edit Script** changes any field into a script. After choosing this menu, a script editor appears. When the code is written and saved, the field is turned into a script. If you close the script window without saving, then the field will not change.
- **Open View Object** opens the object responsible for the view. This is only for view system programmers.
- **Open Display Object** opens the object responsible for this display of the view. This is only for view system programmers.

Menu Bar

The document main window (the window that initially is displayed when you start the document) contains a menu bar with six menus.

Document, Edit, & Option

The first three menus are the standard menus that all PenPoint applications display. They function in the standard way.

To see which version of Glyphic Codeworks is running:

1. **Tap 'About…' from the 'Document' menu.** An option sheet appears.
2. **Turn to the 'About Application' page.**

Utilities

The Utilities menu has various options described elsewhere in this chapter in detail.

- **Palette...** opens the Palette of standard objects that can be copied into views.
- **New Class...** creates a new class and opens it for editing. *See Create New Object, in this chapter*
- **New Workspace...** opens a new workspace window. *See Workspaces, in this chapter.*
- **Folders...** opens the folder browser. If the folder browser is already open in the main document window, then it just flashes the borders.
- **Projects** opens the 'projects' object in a property browser. *See Exploring Projects, in this chapter.*
- **Breakpoints...** finds a list of all scripts that have breakpoints.
- **Implementors of...** finds all objects that implement a given property. A note is displayed for you to enter the name of the property. If you select some text, for example in a script editor, that text will already be in the note.
- **References to...** finds all scripts that reference a given property. This operation may take awhile. A note is displayed for you to enter the name of the property. If you select some text, for example in a script editor, that text will already be in the note.

System

This menu has several useful commands listed below. The remainder are commands that most users will never need. For more information about those commands, *see The System Menu, in chapter 6, Advanced Topics.*

- **Windows** is a sub-menu that contains a list of all open windows for the document, except the document's main window. Tap on any window in the sub-menu to make bring that window to the top.
- **Transcript...** opens the transcript window. Your scripts can write debugging messages to this menu by sending the **put** message to 'transcript'. For example:

```
if (x > 70) then [ transcript put "x is very large" ].
```

- **Breakpoints** is a switch. If on (checked) then breakpoints are enabled. If off, then all breakpoints will be ignored.
- **Power Menus** is a switch. If on (checked) then all menus in sections are shown. This makes choosing menu items quicker because they are all displayed at once and you don't have to tap on section headings to see the menus in them. On the other hand, this makes for very large menus that obscure much of the screen when they're open.

Chapter 2

View Procedures

Working with Views

Changing View Modes

Flick left-right on the title line of a form toggles its mode between author and User modes.

In User mode, the form is a fully functional user interface. You can change the values of fields using the standard techniques of PenPoint. This is the mode where you use the objects you've built.

In Author mode, the form can be built and edited. You can change what fields are on the form, how it looks, and how it interacts with the user. This is the mode where you construct your objects.

Changing View Options

Views have a variety of options that can be changed. Not all views have the same options, see their descriptions later in this manual to learn more.

To open a View Option sheet:

1. **Check on the view.** A pop-up menu of the view's options appears.
2. **Tap on a option.** The option sheet opens.

Adding & Removing Views

All these procedures work on form views. Make sure the form is in Author mode before beginning.

Adding a View

To add a field to a view:

1. **Up-caret in the form where you want the new view.** A pop-up list of view types appears. You must do this in an empty space in the form.
2. **Tap on a view type in the list.** The view is added to the form.

Another way to add a field is to copy or move an object into an empty space in the form. The object may come from either:

- **the Palette**
- **a value in a browser, or workspace**
- **a property from a browser**
- **another form in Author mode**

If the property doesn't exist in the object the form is displaying, then the property is added to the object. To change the property's name, *see Renaming Views later in this chapter.*

To add a view for an existing property in the object:

1. **Open the property browser for the object.**
2. **Press on the property name until the move marquee appears.**
3. **Drag the marquee into an empty space in the form.**

Removing a View

To remove a view and the property it displays:

1. **Cross out the field.**

To remove just the view, leaving the property:

1. **Double tap on the field to get the view menu.**
2. **Tap on Delete View Only.**

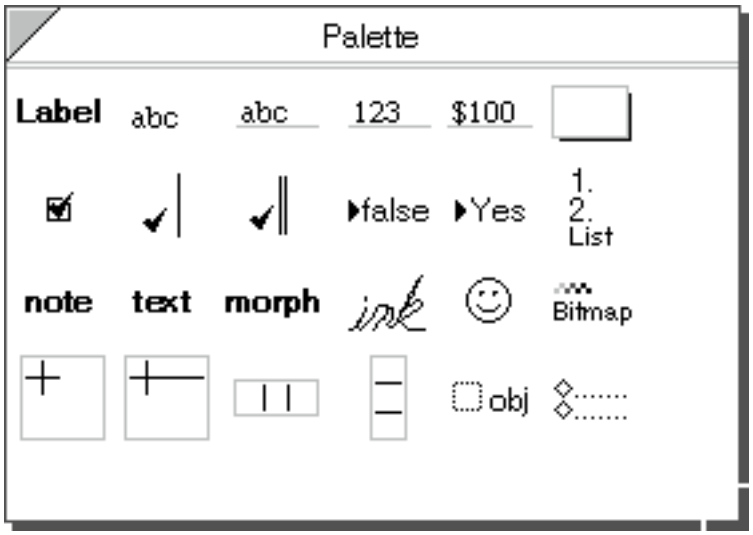
The Palette

The palette contains examples of views and aligners that are useful in building objects. The items on this palette can be copied into the form views of other objects.

Opening the Palette

To open the Palette:

1. **Tap on 'Palette…' in the Utilities menu.** The palette appears:



Items on the Palette

The palette has the following items on it:

Top Row	Label, Read-only String, String, Number, Number with money formatting, Button
Row 2	Checkbox, Choice, Multiple Choice, Pop-up Choice, Pop-up Choice with Yes/No entries, List
Row 3	Note (like MiniNote), Text (like MiniText), Morph, Signature (ink), Face, Bitmap
Bottom Row	several aligners, Object, Browser

Renaming Views

All these procedures work on views in a form. Make sure the form is in Author mode before beginning.

Renaming Properties

Each view in a form (except labels) is a property of the object being displayed in the form. To rename the property:

1. **Circle on the view.**
2. **Edit the name in the note that appears.**
3. **Tap Rename.**

Renaming Labels

Label views in a form do not correspond to a property, they are just text displayed in the form. To change the name displayed in a label:

1. **Circle on the label.**
2. **Edit the name in the edit pad.**
3. **Tap OK.**

Renaming Controls

Controls (check boxes and buttons) display a name in the form. This name is usually the name of the property that the control is for. To rename the property use the procedure above. However, sometimes it is desirable to have a different name displayed.

To change the name displayed in a control, without changing the name of the property:

1. **Double tap on the control to get the view menu.**
2. **Tap on Rename Control in the view menu.**
3. **Edit the name in the note that appears.**
4. **Tap Rename.**

To reset the control back to using the property name:

1. **Double tap on the control to get the view menu.**
2. **Tap on Rename Control.**
3. **Clear the text of the name in the note that appears.**
4. **Tap Rename.**

Aligners

Aligners are grids that determine the layout of a form. Aligners automatically place views in orderly arrangements according to the specifications you set and the size of the user's screen, system fonts, and notebook.

NOTE: Aligners are the powerful tool that allows you to design forms in Glyphic Codeworks. Because of their power they offer a large set of options that can be used in many ways for many different effects. Aligners will be easier to understand if you create a test object and try out the different settings to see their effects on the form.

An aligner specifies a grid of cells, much like a spread sheet. Each cell can contain a view, be empty, or contain another aligner nested within it. For each row and column the aligner determines the width and height, and the placement of the views within.

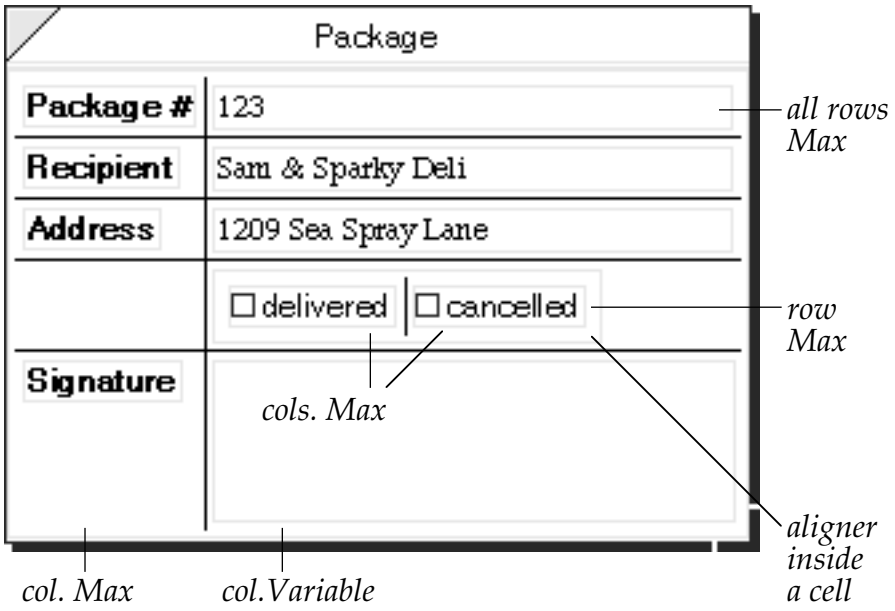
Normally, as you add fields to an aligner, the aligner automatically sets its own options to work with the fields. For example: adding a note field to an aligner will automatically make the row and column the note is in expand to fill the available space in a window.

You can specify exactly how you want to the aligner to work through options for each row and column:

Alignment	determines the placement of views within cells, vertically for rows, horizontally for columns
Min	top / left
Center	center
Max	bottom / right
Fill	the views are expanded in size to fill vertically / horizontally

Extent	determines the size of the cells; adjusts the height for rows, width for columns
Max	size is as big as the biggest view in the row
Fixed	size is constant
Variable	size is a portion of the available space

Extent Value	used with Fixed and Variable size settings
for Fixed	extent is the size in pixels *[this will become device independent, but will be approximately pixels in size]*
for Variable	portion of the available space to take

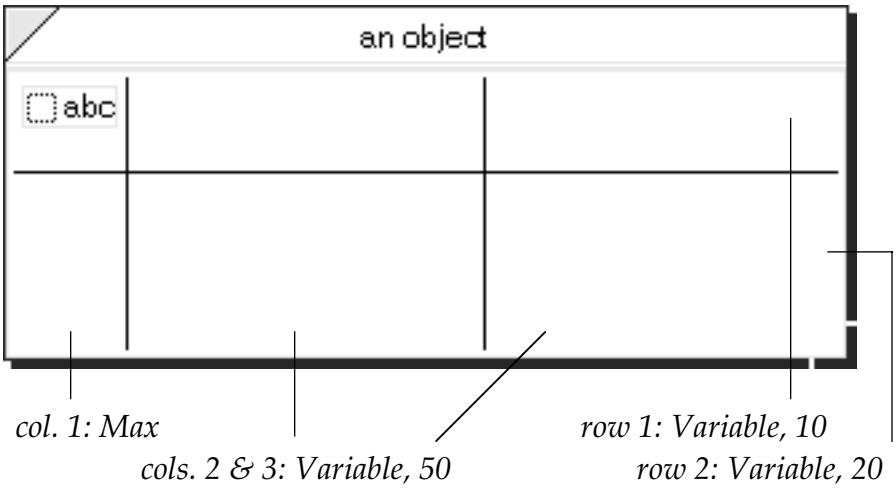


It is common to want a number of rows or columns to have the same settings. To make this easy, there is a default row and a default column set of options for each aligner. If there are no specific settings for a given row or column, then it will use the default row or default column settings.

Variable Extent

When a row or column extent is set to Variable, it divides the available space in the window with the other rows or columns that have Variable extent. They divide the space like portions in a cooking recipe: The extent value determines how many portions each row or column gets. The total of all extent values determines how many portions the available space is divided into.

For example:



The two rows are both Variable, so they share the entire height of the view. Row 1 takes 10 parts, row 2 takes 20: so the height is divided into 30 parts and split between them. The effect is that the upper row is 1/3 of the height and the lower 2/3.

The first column is Max, so it is the width of the widest thing in it. Columns two and three are Variable, and so they share the remaining width of the view. Since both rows take 50 parts, the remaining width is divided into 100 parts, and they each get half.

Notice that if the total number of parts for the width or height is 100, then the variable extent values are percentages. However, the total number of parts can be more or less than 100.

Working with Aligners

The following procedures work on aligners in a form. The form must be in Author mode for these operations to work. When the form is in Author mode, the aligners in the form are drawn with gray edges and black grid lines.

[Sometimes aligners don't resize everything immediately. You can always force an aligner to resize at any time by resizing the window to a slightly different size.]

Changing the Aligner Grid

The number of rows and columns, and the spacing between rows and columns (the gap) can be changed for each aligner. The change is made on the Aligner page of the option sheet for the aligner.

1. **Check on any part of the aligner.** Alternatively you can double tap to get the menu, and then tap on View Option.
2. **Tap on Aligner Options from the pop-up list.**
3. **Turn to the Aligner Size page of the option sheet that appears.** It may already be showing.
4. **Change the number rows and columns as needed.**
5. **Change the sizes of the gaps as needed.**
6. **Tap Apply.** After a pause the aligner is updated to reflect the change.

Changing the Default Row or Column Setting

1. **Check on any part of the aligner.** Alternatively you can double tap to get the menu, and then tap on View Option.
2. **Tap on Aligner Options from the pop-up list.**
3. **Turn to the Aligner Alignment page of the option sheet that appears.** It may already be showing.
4. **Tap on 'default row' or 'default column' in the list at the top of the sheet.**
5. **Change the options at the bottom of the sheet as needed.**
6. **Tap Apply.** After a pause the aligner is updated to reflect the change.

Changing a Specific Row or Column Setting

1. **Check on any part of the aligner.** Alternatively you can double tap to get the menu, and then tap on View Option.
2. **Tap on Aligner Options from the pop-up list.**
3. **Turn to the Aligner Alignment page of the option sheet that appears.** It may already be showing.
4. If the row or column you want to change is not listed: **Caret in the list, then tap on the row or column in the menu that appears.**
5. **Tap on the row or column in the list at the top of the sheet.**
6. **Change the options at the bottom of the sheet as needed.**
7. **Tap Apply.** After a pause the aligner is updated to reflect the change.

Resetting a Row or Column to the Default Setting

1. **Check on any part of the aligner.** Alternatively you can double tap to get the menu, and then tap on View Option.
2. **Tap on Aligner Options from the pop-up list.**
3. **Turn to the Aligner Alignment page of the option sheet that appears.** It may already be showing.
4. **Cross on the row or column in the list at the top of the sheet.** If the row or column is not listed, then it is already using the default setting. After a pause the aligner is updated to reflect the change.

Adding an Aligner into a Form View

If you want to further divide the space of a cell in a form, you can add an aligner grid by either:

1. **Up-caret on an empty space in the aligner.**
2. **Choose Aligner from the pop-up menu.**

or

1. **Open the Palette.**
2. **Tap-Press on one of the aligners to get the copy marquee.**
3. **Drag the marquee into the cell of the form.**

Deleting an Aligner from a Form View

You can delete an aligner embedded within a form:

1. **Move or delete all the views within the aligner.**
2. **Cross on the aligner.**

Making a View the Main View

When you turn to a Glyphic Codeworks document, the main window of the document displays the **folder browser**. You can change this so the main window displays an object you've built that has a view. From then on, that is the object that you'll see every time you open the document.

To set the object that the main window displays:

1. **Open the object in the view you want as the main view.**
2. **Tap on Make Main Win from the title bar menu.** The window will expand and fill the main document window. The object that was the main window (often the folder browser) will float in its own window.

To reset the main window to the folder browser:

1. **Tap on Folders… in the Utilities menu.** The folder browser opens in its own window.
2. **Tap on Make Main Win from the title bar menu of the folder browser.**

Alternate Views

Objects can have more than one view. Each view is a different way of looking at the object. For example, strings can be viewed with either a string view or a read-only string view. Each object has one view designated as its standard view. The standard view is the view for the object that the system uses by default. The other views for an object are called its alternate views.

Choosing which view to use in a form

After you have added a field to a form, you change which view is used to display it. You can choose from any of the object's alternate views.

To choose a view for an field in a form:

1. **Check on the object while the form is in Author mode.** An a pop-up list of options appears.
2. **Tap on View Type Options.** An option sheet appears.
3. **Choose a view from the list.**
3. **Tap Apply & Close.**

Creating an alternate view

To create an alternate view for an object:

1. **Check on the object.** A pop of list of option sheets appears.
2. **Tap View Management from the pop-up menu.** The view management option sheet for the object appears.
3. **Up-caret in the All Views list.** A pop-up list of form types appears.
4. **Tap on one of the form types.** A new form view for the object appears.
5. **Edit the new view.** Dragging items from the standard view is a common way to add views to an alternate view.

Designating a standard view

To change a particular view of an object to be the standard view.

1. **Check on the object.** A pop of list of option sheets appears.
2. **Tap View Management from the pop-up menu.** The view management option sheet for the object appears.
3. **Drag the view from the All Views list to the Standard View field.**

Renaming a view

By default, views are given names like “class-1-view-2”. Since this name is used when you use the Open… menu and in the View Type options sheet, you may want to give a view a more descriptive name.

To rename a view for an object:

1. **Check on the object.** A pop of list of option sheets appears.
2. **Tap View Management from the pop-up menu.** The view management option sheet for the object appears.
3. **Circle on the view in the All Views list.** A pop-up note lets you edit the name.

Removing an alternate views

To remove an alternate view for an object:

1. **Check on the object.** A pop of list of option sheets appears.
2. **Tap View Management from the pop-up menu.** The view management option sheet for the object appears.
3. **Cross on the view in the All Views list.**

Object Menus

Your classes can define their own menus. These menus will appear almost everywhere the other standard object menus, such as Move and Open, appear. When a user chooses the menu entry, a message is sent to the object and a script run. This is very similar to what happens when a button is pressed.

Menu Editor

Menus are edited in a menu editor. You can open a menu editor with the command Edit Menus. The Menu Editor displays a list of all the menus for an object, including those inherited from its parents. Tapping on a menu entry selects it and allows you change its values in the bottom half of the window. You can Up-caret to add a menu and Cross to delete one.

Each entry for a menu specifies several pieces of information:

- **Title:** the string that is shown in the menu.
- **Script:** the message sent when the menu is chosen.
- **Gesture:** if any, a gesture short cut that invokes the menu.
- **Sort Key:** the number used to sort the entry along with the other menu entries.
- **Options, for view:** should not be set, used only in view menus (see below).
- **Options, development only:** if set, this menu will only be shown if you have the development environment. Use this for menus that are only for debugging.

If the sort key is a multiple of a thousand, then the entry is a section header. Section headers are shown and bold, and simply group the menus after them. Tapping on them doesn't send a message.

[There is a known bug with section headers: the Menu Editor expects the Script value to be ??? (not the symbol $???, the value ???) for section headers. There is no way to set the script to that. The section header will still work property, it just won't look like a header in the Menu Editor list.]

The menus in the Menu Editor sort whenever you add or delete one. However, in the course of editing menus, they may become out of order. The Sort Menus button will resort them.

To edit the script that a menu will execute when chosen, select the menu entry and tap the Edit Script button. If the menu is defined in the object being edited, then the script will open up in a script editor. If the menu is defined in a parent then you'll be given the option to either edit the parent's script, or override the script. Once overridden, the Edit Script button will automatically open the object's overriding script.

To override aspects of a menu other than the script, create a menu entry with the same sort key as the menu your overriding. In this way, and object can override the gesture or title of a menu. If object just needs to override the script, it can do this without its own menu entry, see above.

View Menus

View objects have a number of different menu lists that can be edited. For view objects, the Edit Menus command displays a pop-up list of these for you choose from. A Menu Editor will be opened for whichever menu list you choose.

The menus lists are:

- **Object Menus:** the menus described above. Since you will probably never create views of view objects (!), you don't define object menus for views.
- **Title Bar Menus:** these menus are available in the title bar of objects opened using this view. They are added to the object's own object menus, and can override them.
- **View Menus:** these menus are available, in user mode, when the view is embedded in another view.
- **Authoring Menus:** these menus are available, in author mode, when the view is embedded in another view. Since these affect the development environment, you rarely edit these menus.
- **List menus:** these menus are available, in user mode, on list items. These menus only make sense for list views. List views have another command that lets you edit these menus: Edit List Commands.

Menu entries for the last four lists can have the 'for views' option turned on. If it is, then when the menu is picked, the message is sent to the view. If it is off, then when the menu is picked, the message is sent

to the object being viewed.

[There is a known bug with view menus and the Edit Script button: it will always edit scripts for the view object, even if the for-view option is off and the message will be sent to the object being viewed, not the view. In this case, you will have to add the script to the object in a browser.]

Chapter 3

Script Procedures

Editing a Script

Glyphic Script is written using formatted text. The formatting of the text means something in the language. Since formatting is so important, there are several enhancements to the normal PenPoint procedures for formatting text that work in script editors:

Formatting with the Pen

- **Flick right on a word or a selection to make the text bold**
  This style is used for messages and keywords.
- **Flick left on a word or a selection to make the text plain.**
  This style is used for variables and properties.
- **Double flick left on a word or selection to make the text italic.**
  This style is used for comments.
- **In edit pads, words that start with a capital letter will be bold (except in quoted strings) and in lower case when the editing is finished.** Remember that case does not matter in Glyphic Script.

Formatting with the Keyboard

- **Typing words that begin with a capital letter will be inserted bold, and in lower case (except in quoted strings).** Remember that case does not matter in Glyphic Script.
- **Typing the enter key not only starts a new line, but will also automatically indent to the same level as the preceding line.**
  These automatic tabs can be deleted with the backspace key.
- **Typing always replaces the selection.**
- **Typing always takes the format of the character just before the insertion point.**

Saving a Script

After editing a script, it must be saved before it can be used by the system. To save a script that has been changed, tap the Save button in the script editor window. If the script has no errors, then it is compiled and saved.

Until a script is saved, the old version of the script, if any, will still be used by the system. When adding a new script, the property won't be added until the script is saved.

Closing Before Saving

If you close a script editor without saving, a note appears asking you if you want to save the script. The note has three buttons:

- **Save** saves the script before closing the editor. If there was an error the editor isn't closed.
- **Don't Save** closes the editor and forgets any changes you made
- **Cancel** doesn't save or close the editor

Script Errors

If the script has an error, the system cannot compile it. When you try to save the script the system will select all the text from the beginning of the script to the point just before the error was detected. A message is added to the top of the script editor window telling you what the compiler found wrong with the script. Correct the problem and try to save the script again. If all goes well then the error message will be removed.

Remember that sometimes the real problem with the script is before the location where the compiler found an error. Common problems of this sort are missing periods, closing brackets, and closing parentheses.

Be careful when editing after an error has been detected. If you accidentally type a key before setting the insertion point, the typed key will replace the portion of the text selected by the compiler to show you where the error was.

Disabling Part of a Script

Glyphic Script uses the Strike-thru text style to indicate text which is disabled and won't be run. Sometimes it is useful while changing or debugging scripts to simply disable a part of a script rather than remove it. Later on, the text can then be easily enabled or removed as needed.

To disable part of a script:

1. **Open the script if it is not already in a script editor.**
2. **Select the portion of the text to be disabled.**
3. **Check on the selection.** The text option sheet appears. Be sure to do this even if the text option sheet is already visible.
4. **Turn the option sheet to the Character page, if needed.**
5. **Tap on Strike-thru to put a check mark next to it.**
6. **Tap Apply & Close.** The option sheet is removed.
7. **Tap the Save button at the top of the script editor.**

To enable part of a script:

1. **Open the script if it is not already in a script editor.**
2. **Select the portion of the text to be enabled.**
3. **Check on the selection.** The text option sheet appears. Be sure to do this even if the text option sheet is already visible.
4. **Turn the option sheet to the Character page, if needed.**
5. **Tap on Strike-thru to clear the check mark from it.**
6. **Tap Apply & Close.** The option sheet is removed.
7. **Tap the Save button at the top of the script editor.**

Script Assistance

Overriding a Script

When you want an object to respond to a message differently than its parent does, you want to override a script. There are two easy ways to override a script from the property browser of an object:

If the script is commonly overridden, it will be in the Override menu in the property browser of the object. To override it:

1. **From the Override menu, tap on the message you want to override.** A script editor will open for that message with a sample script with comments. The Override menu is available both from the top of a property browser, and from the pop-up menu for the object in the Object Development section.
2. **Edit the text and tap Save.**

To override a script:

1. **Locate the script in the objects parent (or grandparent).** You may need to double-tap on the **parent** property at the top of the property list to open the parent object.
2. **Press on the left side of the script until the move marquee appears.**
3. **Drag the marquee to the property browser of the object you want the override for.** A script editor will open with a copy of the parent's script.
4. **Edit the text and tap Save.**

Using Templates

A template is a piece of a text that can be inserted into a script. At the top of a script editor there is a Templates menu.

To insert a template:

1. **Set the insertion point in the script.**
2. **Tap on one of the templates listed in the Templates menu.** The text of the template is inserted
3. **Edit the inserted template to your needs.**

Dragging Names

As a short cut for typing any property or object can be dragged into a script. The name of the property or object is inserted into the script text where you dropped the move marquee. The inserted name is always

in plain formatting. You can flick right on the name to make it bold.

Cross Referencing

The system has some tools for finding out about the usage of scripts and objects throughout the system. The result of these queries is a group of scripts. Tap on entries in the group to see the scripts.

Implementors of a Property

Implementors of a property are those objects that have that particular property. This operation takes a few seconds to execute. The operation can be executed three ways:

From a property browser:

1. **Double-tap on a property name.** The property pop-up menu appears.
2. **Tap on Implementors.** After a delay a group listing all implementors of that property appears.

From a script editor:

1. **Double-tap on a title bar.** The property pop-up menu appears.
2. **Tap on Implementors.** After a delay a group listing all implementors of that script appears.

From the Utilities menu:

1. **Tap on Implementors Of…in the Utilities menu**. A note appears.
2. **Enter the name of the property.** If you select some text (for example, in a script editor) before you do step 1., then that text will already be entered in the note.
3. **Tap OK.** After a delay a group listing all implementors of that property appears.

References to a Property

References to a property are those scripts that use the property. This operation can take up to a full minute to execute because the system must check every script one by one. There are three ways to do this operation:

From a property browser:

1. **Double-tap on a property name.** The property pop-up menu appears.
2. **Tap on References.** After a delay a group listing all references to that property appears.

From a script editor:

1. **Double-tap on a title bar.** The property pop-up menu appears.
2. **Tap on References.** After a delay a group listing all references to that script appears.

From the Utilities menu:

1. **Tap on References To…in the Utilities menu**. A note appears.
2. **Enter the name of the property.** If you select some text (for example, in a script editor) before you do step 1., then that text will already be entered in the note.
3. **Tap OK.** After a delay a group listing all references to that property appears.

Instances of an Object

To find all objects whose parent is a given object, for example **folder**, use the script:

```
browser instances of folder; open
```

Replace 'folder' with what ever parent whose direct children you want to find. Note: this does not find objects that inherit indirectly from **object**. This operation can take from just a few seconds to up to a minute depending on how many instances are found.

Chapter 4

A Catalog of Views

Textual Views

A number of views display as a small piece of text:

- **String Views** display a editable string.
- **Number Views** display a editable number.
- **Read-only String Views** display a string, number or the name of any object. They are not user editable.
- **Label Views** display a fixed string. Label views are used for decorating a form.

Font Options

You can change the font, size and style of Read-Only and Label views. These options are edited on the Font Options option sheet for the view.

In the Font Options option sheet there are choices for font, size and style. The font & size choices have a value choice 'default' use what the user has chosen as the PenPoint Font and Font Size in the Settings notebook. The other font size choices are actually relative to the Settings notebook: If you choose 12 it will be the user setting; if you choose 14 it will be 16.67% larger; if 10 then 16.67% smaller; etc.

Number Formatting

A number displayed in a Read-Only or Number view is formatted with a number format string. This formatting string is edited on the Number Format option sheet for the view.

In the Number Format option sheet you can choose one of the standard formats from the check list, or you can write you own format string in the write-in field. See the documentation for the **scan** message in the Language manual for details about number format strings. The top choice in the standard formats list is empty: A blank string will display numbers in the normal manner.

Label Views

Unlike the other three, label views do not correspond to any property of the object displayed by the form. Instead, they present static text on

the form. To edit this text, Circle on the label in author mode.

Validated String Views

String views can be validated. There are three ways that a string view can be validated: word lists, character lists, and a set-value script. Each method restricts the values that can be written into a string view in different ways. The different techniques can be combined. For example, number views are actually string views that have both a character list and a set-value script.

Word Lists

A word list is a list of strings that are possible valid values for the string. When the user writes in the string view, PenPoint uses the list of strings to assist in translation. For example, if the list of strings is the list of US states, then writing 'Cal' will translate to 'California'.

[PenPoint's implementation of word lists for fields (the basis for string views) is incomplete. It will allow the user to type anything on the keyboard, and will at times give up on the translation and let non-valid strings into the field. In this release, string view does not force the final result to be a choice from the word list. It may do so in the future.]

To edit the word list for a string view:

1. **Open the View Type Options for the view.**
2. **Change the view to a pop-up choice view.** This change is only temporary.
3. **Edit the choices of the pop-up to be the word list**. Be sure to have only strings in the list.
4. **Open the View Type Options for the view.**
5. **Change the view back to a string view.**

Character Lists

A character list restricts the characters that may be entered into a text field. When the user writes in the string view, PenPoint uses the list of characters to assist translation and restrict character entry to just those characters in the list.

[PenPoint's implementation of character lists for fields is incomplete. It will allow the user to type characters not in the character list into the input pad. In this release, string view does not restrict the final result to be from the char-

acter list. It may do so in the future.]

To edit the character list for a string view:

1. **Double tap on the string view.**
2. **Tap on Open View Object from the view menu.** A property browser opens.
3. **If needed, add a property called char-list.** The property should be a class property.

4a. Edit the property char-list to be a string with the allowable characters for the string view.

-or-

4b. Delete the property char-list to remove any restrictions and allow all characters.

Set-Value Script

A set-value script allows you to make the string view interpret and control the possible values of the string view any way you want. Every time the user has finished editing the string the user's string is passed as the argument the message set-value. The standard implementation saves the string in the object that is being viewed. A custom script could change the string in some way, or even replace it with some other object.

An example set-value script might be:

```
$ to new-value, display d.
$ fixed-value.
fixed-value := new-value.                          get the user's string
if (".!?" has fixed-value.last; not )              does it end in punctuation?
      then [ fixed-value := fixed-value & "." ].   if not, add a period
super set-value to fixed-value display d.          send to super to set it
```

To add a set-value script to a string view:

1. **Double tap on the string view.**
2. **Tap on Open View Object from the view menu.**
3. **Add a global script property called set-value.**
4. **Edit the script.** Ensure the script takes the arguments 'to' and 'display', and sends set-value to super with the corrected value. See the example above.
5. **Tap Save.**

Control Views

Buttons and Checkbox Views are controls.

A button is a control for a script. When the button is pressed, the script is executed. When you tap on a button in author mode, a script editor opens so you can edit and save the script.

A checkbox is a view of a boolean value (true or false). Tapping it toggles it between true and false.

Control Strings

Controls display a string: inside the button, and next to the checkbox. By default, this text is just the name of the property. You can change text to be anything you want. The text font, size and style can also be set in the Font Options option sheet for the view. For more information about Font Options, *see Textual Views earlier in the chapter.*

To change the text:

1. **Double tap on the checkbox view to get the view menu.**
2. **Tap Rename Control from the view menu.**
3. **Edit the name in the note that appears.**
4. **Tap Rename.**

To restore the text to be the property name:

1. **Double tap on the checkbox view to get the view menu.**
2. **Tap Rename Control from the view menu.**
3. **Clear the name in the note that appears.**
4. **Tap Rename.**

List View

List views display a group in a list box. The items of the group are displayed one per line. The string for each line is the item's string.

List Menus

In user mode, there is a pop-up menu available. This menu is composed of the menus of the item under the gesture and the list's menus. You can edit the list menus for a particular list view inside a form.

To edit the list menus for a list view:

1. **In author mode, double tap on the field.** The view menu appears.
2. **Tap on Edit List Commands.** The menu editor appears. *[This should be named Edit List Menus].*

From the Menu Editor, you can either override the effects of a menu, or add new menus. These operations add scripts to the list view object, not the class you are building. See the sample script you get when you override or add a menu for details of how list commands get access to the object being displayed, the item the gesture was own, and the group displayed in the list view. *See Editing Menus in View Procedures for more details on the Menu Editor.*

Choice Views

There are three kinds of choice views:

- **Single Choice** views display a list of choices with a single line on the left and a check mark next to the current choice. The value of the property is set the current choice.
- **Pop-up Choices** display the current choice with a triangle to the left. Tapping on the choice displays a pop-up list of choices. The value of the property for the view is set to the current choice.
- **Multiple Choice** views display a list of choices with a double line on the left and check marks next to all currently chosen choices. The value of the view's property is set a group of all currently chosen choices.

[In future releases, PenPoint may no longer support Single and Multiple Choice views.]

Editing the Available Choices

The possible choices for a choice view are held in a group. The group can have either strings, numbers, or entries. An entry specifies the string or bitmap to be displayed in the choice view, and the value to be used for the property. Choices are edited in a Choice Editor.

To open a choice editor for a choice view:

1. **Double tap on the choice view to get the view menu.**
2. **Tap Edit Choices from the view menu.** The Choice Editor appears.

The following procedure all take place in an open choice editor.

To add a string or number choice:

1. **Tap on String or Number in the Insert row.** A new choice appears at the top of the choice list and an expression editor opens up for editing the value.
2. **Edit the value in the expression editor.** If the value is a string be sure to keep the quotes around it.
3. **Tap Save and close the expression editor.**

To add an arbitrary object:

1. **Drag the object into the choice editor's list.** The object will be added to the choice list. In the choice view, it will display its name.

To add entry:

1. **Tap on Entry in the Insert row.** A new entry choice appears at the top of the choice list and the details are in the bottom of the editor. At this point you can:

- **Edit the string.** This is the value displayed in the choice view.
- **Edit the value.** Follow the next procedure.
- **Change the bitmap.** Drag a bitmap editor document, or a bitmap object to the bitmap field.

To edit a string or number choice, or the value of an entry:

1. **Tap on the choice.** The choice selects.
2. **Tap edit-value.** An expression editor opens.
3. **Edit the value, tap Save, and close the expression editor.**

[A known bug is that if the choice is visible in another window, it may not reflect some edits immediately. Tapping another choice entry to change the selection in the Choice Editor will cause the choice to update.]

To delete a choice:

1. **Cross on the choice.**

To reorder the choices in the choice editor:

1. **Move the items around in the list.** *[A known problem is that the items end up in the list twice. To fix this, simply delete the choice where it came from after you move it to a new place in the list.]*

Component Views

Component views display a standard PenPoint component. There are three component views:

- **Note View** is based on the MiniNote component. It displays and edits ink and translated text. The view has a gesture margin and a scroll margin and responds to the full range of MiniNote commands. The data stored in the view's property is an instance of PenPoint's clsNotePaper.

- **Text View** is based on the MiniText component. It displays and edits text. The data stored in the view's property is a string. As a consequence, the text cannot be formatted (ASCII strings have no way of storing formatting information).

- **Ink View** is based on PenPoint's clsScribble. It displays and edits ink. Only the scratch out gesture is available. The data stored in the view's property is a scribble object.

Bitmaps

Bitmaps can be used to decorate a number of other views: labels, buttons, and choice entries. Bitmaps can also be a data type used in your classes.

Bitmaps are not edited in the Glyphic Codeworks environment. They are instead edited with the Bitmap application that comes with the PenPoint SDK. You should install that application on your machine.

To set the bitmap of a label or a button from a Bitmap document:

1. **Put the view in author mode.**
2. **If the bitmap document is open, either close it or checkpoint it.** Checkpoint is in the Document menu.
3. **Copy the bitmap document onto the label or button.** Be sure to copy it, if you move it, the document will de deleted after the operation.

To set the bitmap of a choice entry from a Bitmap document:

1. **Open a Choice Editor for the view.**
2. **Tap on the entry to select it.** The entry will appear in the bottom of the Choice Editor.
3. **If the bitmap document is open, either close it or checkpoint it.** Checkpoint is in the Document menu.
4. **Copy the bitmap document onto Bitmap field in the bottom of the Choice Editor.** Be sure to copy it, if you move it, the document will de deleted after the operation.

Bitmap View

A Bitmap document can be dragged in the same manner into any property in a browser or onto an object view. This lets you set the value of object's property to a bitmap object. These bitmap objects can be dragged onto labels, buttons and choice entries as well.

A bitmap view is a view of a bitmap object. It simply displays the bitmap and is not editable. Bitmap views are useful when they are viewing a property that is a script that returns one of several bitmap objects (stored in other properties or a group) to show the state of the object.

Morph Views

Morph Views display the standard (or some other) view of the value of its property.

Normally, when you embed one object you've build, for example a film, into another, say a film catalog, you create a property in the film catalog with a view suitable for films. You can only set that property to film objects, because film catalog has a film view in it. If you put any other kind of object, perhaps a film table of contents, into that property of the film catalog, then the view will fail to display correctly. However, if you use a morph view, then the view will automatically change to accommodate whatever object is placed in the property: switching between the film view and the film index's view.

You can control what views the morph view uses. Open the view object for the morph view copy the script default-view from its parent (morph view) to your view. Then follow the comments in the script to write your own.

Views for Classes

When you create a class, and when you add a new view for a class, you choose what kind of view to start with. In both cases there are four choices: Labeled From, Split Form, Control Panel, and Drawable View. If you are creating a class, these are show iconically, if you are adding a new view, they are listed in a pop-up.

The first three are Form Views preset with various aligners and options. A Form View always contains at least one aligner (that can't be deleted) and may have other aligners embedded in it. As you add and edit views to these aligners you are adding and editing the properties of the class. This is the common way of constructing views for your classes. *See Working with Aligners, in the View Procedures section of this manual.*

The Drawable View is a blank area where scripts you write can draw a graphic representation of a class. Once you have created a drawing view for an object, when you use that object in another class, you can choose to have displayed with your graphical drawing view.

Drawing in a Drawable View

When you create a drawable view, you have to write a script that draws the graphic rendition of your object. This is done by sending messages to the 'canvas' object. Canvas understands a number of messages for drawing and setting up the drawing environment. Browse the scripts of the canvas object (in the System Drawing folder) for more details about its messages. Examples of drawing scripts can be found in the default draw script for drawable views and in the Mancala sample application.

To edit the draw script for a drawable view:

1. **Open the object into the drawable view.**
2. **Double tap on the title bar.** The title bar menu appears.
3. **Tap on Edit Draw Script.** A script editor appears. If this is the first time you've edited this script, it will have a sample drawing script. Tap Save to see it work.

Object View

Object views display any object as an icon followed by the name of the object. The icon is based on the object's type. They are useful for having a reference to any other object. Object views operate the same as the right hand side of property browsers.

Browser View

A browser view displays an object in a property browser. However, since browsers are not available at run-time, they are of limited use.

Chapter 5

Debugging

Interrupting A Running Script

You can interrupt running scripts. You may want to do this if you think a script is running too long due to a programming error.

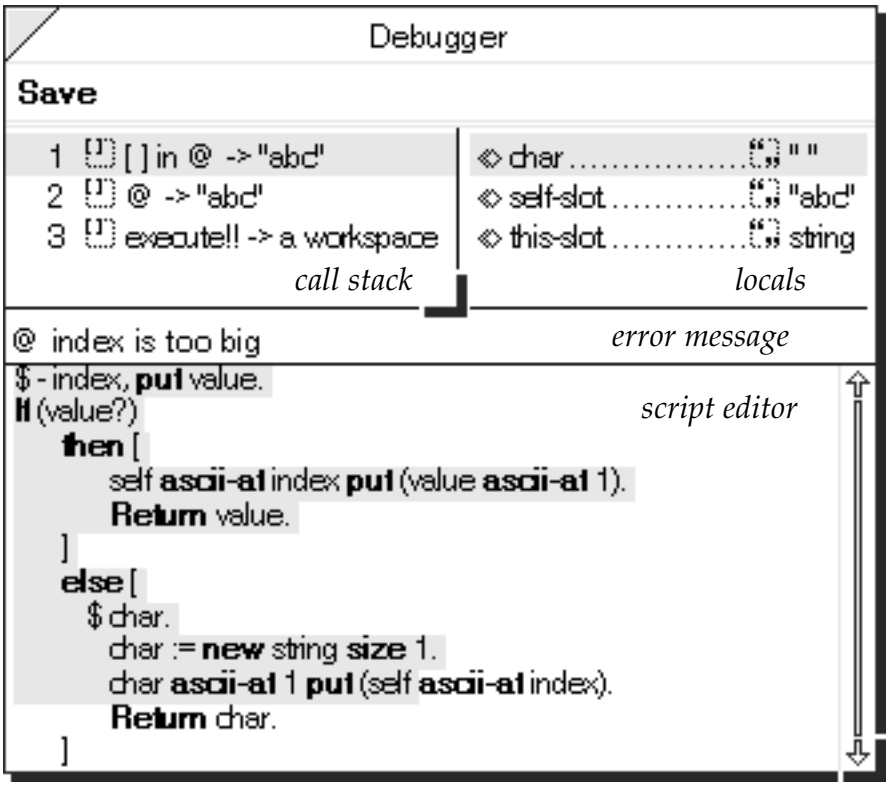
To stop a running script:

1. **Press the pen in the lower left hand corner of the screen.** If a script isn't running, this will probably bring up the move marquee for whatever icon is on the left of the bookshelf. In that case, simply tap on the marquee to dismiss it.
2. **Hold the pen down until the note appears. Then lift the pen.** This takes about a second and a half.
3. **Tap Halt.** After a pause, a debugger window appears.

This operation stops the script in the middle of whatever it was doing. It cannot undo any work that the script has already performed. Be aware that the script has not finished and may have left some objects with intermediate values.

Using the Debugger

A debugger appears when there is an error in a script, or the script was interrupted. Debuggers have four main areas:



The Call Stack

The call stack is in the upper left area of the debugger. It displays the state of the script when it stopped. Each item in the list was called by a message from the item below it. At the top of the list is the script that was running when the error occurred. At the bottom of the list is the message that started the script running.

Each entry in the stack is of the form ‘m -> o’ where ‘m’ is the message that was sent and ‘o’ is the name of object to which the message was sent. Some entries are of the form ‘[ ] in m -> o’. These are entries for blocks inside a script.

Tap an entry in the call stack to set the script editor and locals sections of the debugger to display that part of the script.

The Script Editor

The script editor takes up the lower half of the debugger. It displays the text of the script for the chosen call stack entry.

When a call stack entry is Tapped, the script editor changes to display the script for it. The debugger also selects all the text from the beginning of the script to where the script was stopped. Often this is just before the message name of the next higher entry in the call stack.

You can edit the script in the editor. It works just like other script editors, *see chapter 3, Script Procedures.*

[Note: you cannot actually change a script that came from a workspace.]

The Locals View

The upper right area of the debugger displays local variables and arguments for the current chosen call stack entry. This works just like a property browser, *see Opening and Browsing Objects, chapter 1, Base Procedures.* Tap on any variable to open up its value.

The locals view also includes two values which aren't variables: The value of **self-slot** is the object that was sent the message. It is same as the script's variable **self**. The value of **this-slot** is the object where the script was found. It is an ancestor of the object that was sent the message. It is the script's variable **this**.

Sometimes an argument wasn't passed a value, or a local was not assigned a value. In this case, the debugger will show **undefined** as the value. You can move this object to other locals to make them undefined, but if you move it to other browsers it will not work: You cannot set properties to be undefined. Only arguments and local variables can be undefined.

The Error Message

In the middle of the debugger view is an error message. This is an indication of why the script stopped running. Usually this message refers to a problem with the script of the first (top-most) entry of the call stack. Tap on the first entry to see the script. Sometimes the error refers to a problem in the second entry of the call stack.

Remember that while the debugger stops exactly where there was an error, the actual problem might be elsewhere on the call stack, or somewhere else entirely. For example: running the following script in a workspace:

```
$ s.
s := new group.
return s @ 5.
```

will result in a debugger appearing. The debugger will have the message 'index out of range', and the top-most entry of the call stack will be 'check -> a group, 0 items ' The error is that the system couldn't get the fifth item of an empty group. However, there is nothing wrong with the scripts of the first two entries of the call stack. The problem is that the script run, (the third entry on the stack), set **s** to an empty group and didn't add anything to it.

When saving a script in the script editor, the error message will disappear if there were no errors in saving, or change to display an error with the script. To restore the original error message, tap on the first entry in the call stack.

Breakpoints

A breakpoint is a marker you can put in a script where the system will stop and open a debugger. It allows you observe the inner workings of a script.

When the system executes a script statement that starts with a breakpoint, it halts execution and opens a debugger. The debugger shows the code with the breakpoint selected. At that point you can use the debugger to look at the script in operation; *see Using The Debugger, and Single Step.*

Adding a Breakpoint

A breakpoint can be placed before any statement in a script. A statement is one or more assignments followed by one or more messages followed by a period (remember that the last statement in a block needn't have a final period).

To add a breakpoint to a script:

1. **Set the insertion point where you want the breakpoint.**
2. **Tap on 'breakpoint' in the Templates menu.** The breakpoint symbol, '§', will be inserted into the text.
3. **Tap on the Save button.** If there is a problem and the breakpoint is selected, you probably cannot set a breakpoint where you tried to.

In order for the breakpoint to work, breakpoints must be enabled. If you add a breakpoint, but the script doesn't stop, breakpoints are probably disabled.

To enable breakpoints:

1. **Tap on the System menu.** The menu appears.
2. **If the Breakpoints choice is not checked, tap on it.**

Removing a Breakpoint

To remove a breakpoint from a script:

1. **Cross on the breakpoint symbol, '§'.**
2. **Tap on the Save button.**

Continuing after a Breakpoint

When a breakpoint is encountered while running a script, the system stops execution, beeps, and opens a debugger.

To continue running the script:

1. **Tap on the first item in the call stack.** The script editor will display the script with the breakpoint.
2. **Tap on the Resume button.** Execution of the script will continue. Note that the debugger will not close until the script finishes.

Finding all Breakpoints

When you are done debugging your scripts, you often want to remove all the breakpoints you set. Since it is often hard to remember where you set them, the system provides a way to find all scripts with breakpoints.

To find all scripts that have breakpoints:

1. **Tap on Breakpoints… in the Utilities menu.** After a short pause, a list of scripts appear.
2. **Tap on a script in the list.** The script opens.

Disabling and Enabling all Breakpoints

Breakpoints can be universally disabled. When they are disabled, scripts with breakpoints will execute just as if they didn't have them.

In the System menu, there is a choice entry called Breakpoints. If this choice is checked, then breakpoints are enabled. If it is not checked, then breakpoints are disabled and the system will ignore them.

Single Stepping

When a debugger appears because of either a user halt or a breakpoint, the script can be 'single stepped'. When the code is single stepped, a small part of the script is run and debugger updated to show the new state of the script. This allows you to see the effect the script has on local variables and other objects as each part of the script executes.

There are four ways a script can be stepped. Each runs a different amount of the script before stopping. In the debugger, there is a button for each type of step:

- **Step** runs the next complete statement of the script and stops. This is the most common way to single step a script.
- **Through** runs through the next message that the script sends and stops after it. In general, this will run a smaller amount of code then step.
- **In** runs the next message the script sends, and then stops inside that script.
- **Out** runs the remainder of the script, returns out of it, and stops at the calling script.

No matter which step button is used, the script will always stop at a breakpoint if breakpoints are enabled, and just after the script returns a value.

In and Out work together to allow you to easily see what arguments a script will send to its next message. Tapping In will run the script a little and stop just at the beginning of the script for the next message. You can examine the arguments in the local variable area of the debugger. Tapping Out at that point will execute the messages script at stop in the original script just after the message returns.

Through is just like tapping In, followed by Out.

Resuming a Script

At any point, you can continue running the script without further stepping by tapping the Resume button. The debugger window will not go away until the whole set of scripts finishes running. If the system encounters a breakpoint, it will stop again.

Chapter 6

Advanced Topics

Importing and Exporting Objects

Glyphic Objects can be saved and restored separately from PenPoint documents. Glyphic Objects are stored in Glyphic Script Object Files which can be transmitted to and from systems that don't understand PenPoint files (such as DOS).

Exporting an Object

To export an object to a file:

1. **Open the Connections application and turn to where you want to put the file.**
2. **Press on the object.** The move marquee appears.
3. **Drag the marquee to the Connections notebook.** The Export note appears.
4. **Tap Glyphic Script Object File as the format to export.**
5. **Edit the name as needed.** It is suggested that Glyphic Script Object Files have a name that ends in the extension '.pro'.
6. **Tap Copy.**

All objects that the object being filed references are filed as well. However, those objects that are homed in some other object, are saved only as external references.

Importing an Object

To import an object from a file:

1. **Open the Glyphic Codeworks document where you want to put the objects.**
2. **Open the Connections application and find the file.**
3. **Tap-Press on the object file.** The copy marquee appears.
4. **Drag the marquee over any part of a Glyphic Codeworks document that can accept drag & drop (browsers, object views, and lists).** The file is read and the object is opened in its own window.

When a file is read in this way, all objects that the object referenced are also filed in. In addition, any external references to objects are reconnected, if possible. Any unresolved objects are left as external reference objects. If these objects are later imported, the external references will be reconnected at that time.

You can find all external references that didn't get connected by running the following script in a workspace:

```
browser get-unresolved .
```

[This incorrectly reports two additional objects which are in fact resolved: the root-project and an array of twenty items in the root-project. Ignore them.]

Glyphic Script Object Files and PenPoint Documents

Glyphic Codeworks saves the objects of your document in a file called **doc.pro** in the PenPoint directory for the document. The main object of this file is **doc-project**.

If you ever encounter a Glyphic Codeworks document which PenPoint refuses to recognize, you can recover the objects by finding the **doc.pro** file and importing it as described above. You may have to use DOS to find the file and copy it somewhere where PenPoint's Connections application will see it, such as the root ('\') directory.

User Primitives

User primitives are C programs that you can call from Glyphic Script. This section gives the procedures for building and installing user primitives. It assumes a high level of familiarity with Glyphic Script, Glyphic Codeworks, C Programming, and the PenPoint SDK. All operations here also assume you are running the PenPoint SDK system on a DOS based computer.

[The information presented here will change significantly in future releases to accommodate how PenPoint will handle installation. However, any C programs for user primitives will still run with only minor modifications.]

Building the User Primitives DLL

User Primitives are compiled into a single DLL named **userprim.dll** This DLL must reside in the Glyphic Codeworks application directory at install time. Each time a Glyphic Codeworks document opens, it calls into this DLL to have it add its primitives to the system.

The **userprim** directory on the distribution disk contains source files for building the user primitive DLL. Copy and modify these files to add your primitives. The contents of this directory are:

- **dll.lbc** describes the entry point, no need to modify
- **fileprim.c** is an example set of primitives that provide an interface for PenPoint's clsFileHandle
- **makefile** is a MKS Toolkit makefile for building the DLL. If you use a different make utility, you may need to convert this file.
- **primmain.c** is the main routine for the dll. You will need to modify this file to call your added routines. See the comments inside the file.
- **xxxprim.c** is another example set of primitives, but much smaller. This is designed for copying and modifying into your own user primitives.

The 'user-prim-project'

You will have to add objects and scripts to interface to your user primitive. You can either add these objects to a document, to the **user-prim-project**, or to your own project. In the later two cases, *see Modifying System Projects, in this chapter.*

Modifying System Projects

[This section is an interim procedure. The final system will probably not allow direct modification of system projects. There will instead be a procedure for updating of system projects.]

If you are writing user primitives, you will need to know how system projects are loaded so that you can modify or add to them. This information is not needed by most users.

How System Projects are Loaded

When a document is opened, two directories are checked for the presence of the system projects: first \\BOOTVOL\GLYPHIC, then the Glyphic Codeworks application's MISC directory. If the system projects are found then they are loaded from that directory.

How to Save a System Project

In the course of running the system, if the system projects are modified, their icon in the **projects** window will changed to filled. System projects are not normally saved when the document closes. If the document is closed, changes to system projects are lost. You have to save a system project manually.

System projects are saved to the directory \\BOOTVOL\GLYPHIC. You must have a complete copy of the system projects in this directory before you can save a system project. To do this you need to have the B800 debugging flag set on your machine so you can access the directory structure. Then, copy the entire directory \\BOOTVOL\penpoint\sys\APP\Glyphic Codeworks\MISC to the top level and rename it GLYPHIC.

To save a changed system project:

1. **Open the project from the 'projects' windows.**
2. **Tap the Save button at the top of the project's property browser.**
3. **Edit the file name to save the project in and tap OK.**
4. **After the file saves, a note appears letting you know where it was saved and letting you enter a version comment.** This comment is stored to a file named the same as the project but with the extension .LOG.

The System Menu

[These three entries in the system menu will not be in the final product.]

Some of these options print debugging information on to the system log. If you are running the SDK version of PenPoint, these will be on the second display, or visible on the main display when you enter the debugger. If you are running on a tablet computer, this debugging output can be displayed with the System Log application in Accessories.

- **Sync Host Views** attempts to update all open views. The first time the system encounters an error while a view is displaying, it will open a debugger. After that, future view system errors will cause a beep, but no debugger. After you fix the script that caused the error, choose Sync Host Views to try out the script. This also has the side effect of letting the system put up a debugger on the next view system error.

- **Garbage Collect** forces a garbage collection. Make sure that all temporary windows and objects have been closed before you do this. After this operation all unused objects are destroyed, and anything left is marked as long term. This can adversely affect performance since all remaining temporary objects are marked as long term.

- **Verify Objects** runs a check on all objects and prints the results.

Index