Glyphic Codeworks the Pre-Alpha Release

Glyphic Technology

Friday, January 28, 1994

This release is the first release of Codeworks for PenPoint 3.0 (Amstel). It is very close to the final product we expect to ship. Almost all of the features that were missing in the Demo III release are now implemented. We are providing this release so that EO can evaluate it and make any last design suggestions before the alpha release next month.

New Feature Highlights

- **Run-time and Developer Versions**
  The product is now shipped as a run-time version and a development version. The run-time version is the only thing needed to run applications developed with the development version.

- **Installable Projects and User Primitives**
  User library projects written in Glyphic Script can now be installed onto a PenPoint system by dragging on to the System Notebook. User projects can include primitives written in C (with the SDK). User installed projects can be used in any Codeworks document.

- **Import / Export**
  Codeworks applications can now import and export files. These applications follow the PenPoint import/export UI guidelines. There are new stream and file objects for reading and writing files with special messages for reading text files .

- **PenPoint 3.0 Interface**
  The user interface has been revamped to conform to the new guidelines. Both development environment and user defined commands are available via the Coach interface. Icons for common menus and mode indication have been added.

Release Notes

We have been testing on the Intel-based SDK version of Amstel. We have had a running version of PenPoint (36j) for the 440 for only a few days and so have not had much time to test the Hobbit version. We don't anticipate any problems due to different architectures, but we do wish we had a longer time to test. Amstel is itself also a source of concern: We have had trouble with both the interface and crashes, and feel that we were unable to work with this version of Codeworks to the level of satisfaction of the previous release because of these troubles. The run-time version of Codeworks should run on a Loki unit, however, we didn't have a unit to test with.

This release is compatible with Glyphic Codeworks documents created with or updated to the Demo III release under PenPoint 1.0. It is not compatible with documents created in earlier releases.

Contents

There are two disks and this document in this release. The disks are a complete replacement for previous versions. The documentation augments the Demo III documentation.

Run-time Disk:

- **Glyphic Codeworks (Runtime)**, a PenPoint 3.0 application ready to install on Hobbit based computers. This is the run-time version of the system.

- **Codeworks Sampler**, a PenPoint notebook on the bookshelf of the disk. This notebook contains a completed and enhanced version of the tutorial application, two games, a several other sample applications.

Development Disk

- **'readme'** is a DOS text file containing a directory listing of the disk.

- **Glyphic Codeworks (Development)**, a PenPoint 3.0 application ready to install on Hobbit based computers. This is the development version of the system.

- **Development Files** for creating user primitives for the system with C and the PenPoint SDK.

Known Problems

There are several known problems with this release. Many of these are due to, or cannot be debugged because of, problems in the current release of Amstel.

- The Runtime version is about 150k larger than it will be.
- User Interrupt of scripts doesn't work.
- Opening the Transcript doesn't work and is disabled.
- You cannot export from an imported Codeworks document.
- The stationary is in the wrong place in the Settings Notebook.
- Locking of applications is still missing functionality.

Welcome

If you have not used the system before, we highly recommend following the tutorial. It will introduce you to many of the features and procedures of the system. We believe that Glyphic Codeworks significantly eases the task of creating many kinds of PenPoint applications. However, Glyphic Codeworks is a rich development environment with many paths, as well as a few potential pitfalls, so we hope the tutorial will start you off in the right direction.

We are interested in hearing about problems and successes with the system and its documentation. Please tell us about how you've used the system.

Welcome to Glyphic Codeworks.

—Glyphic Technology

http://www.glyphic.com/

Manual and Disk contents
Copyright © 1992-1993 Glyphic Technology.
All Rights Reserved.

Glyphic and Codeworks are trademarks of Glyphic Technology.
All other products or services mentioned in this document are identified by trademarks or service marks of their respective companies.

Full Change List

This section was prepared from reports generated by our source control system (MKS' RCS software), and our bug database (HyperCard based). The reports were reorganized by functional area.

Object System

- User installable projects and primitives
- Sped up reading in projects
- Primitive tables expand as needed
- Improved primitive hash function
- GObjType() returns gobjTypeSymbol & gobjTypeString
- Changed GObjAddPrim to GObjAddPrimGroup
- Registering fast semaphores as per Amstel

Language

- Duplicate keywords in a declaration signal a compiler error
- Stream and File objects
- Fixed conversion of strings to symbols
- Projects with no exports import correctly
- Change bit added to slot add/remove, array & byte access
- Unicode support in string objects (not complete)

Codeworks

- Port to Amstel
- Division into Run-time and development versions
- Menus available from the coach
- Import & Export support
- Reference buttons (via dragging a document into an alinger)
- Unicode support in labels, fields, and controls
- Options available from coach customize section
- Quick help for labels & aligners
- Labels have no menus

Development

- Common menus available from title bar icons
- User/Author Mode icon in title bar
- Better synchronization of browsers
- Checkpoint and Exit Without Saving commands
- Project manager control panel
- Import & Export control panels

Run-time & Development Applications

Codeworks comes in two versions: Run-time and Development. The run-time version is needed to run any application developed with Codeworks. The development version is needed to create and edit applications.

Installation

To install Codeworks on your PenPoint computer:

1. **Drag the "Glyphic Codeworks (Runtime)" document from the distribution media onto the Settings Notebook on your main bookshelf.** The system will display progress notes during the installation.
2. **Copy the "Samples Notebook" from the distribution media onto the main bookshelf.**

   [In this release, depending on the media you received, these documents may be in the Exchange folder of the distribution media. In the future we hope to have all installation documents o n the bookshelf of the media.]

If you want to edit or create your own Codeworks applications, you will need to install the development version. The development version requires the run-time version to be installed as well.

To install the development version of Codeworks:

1. **Install the run-time version of Codeworks, following the procedure above.**
2. **Drag the "Glyphic Codeworks (Development)" document from the distribution media onto the Settings Notebook on your main bookshelf.** The system will display progress notes during the installation.

Once installed, when you turn to a Codeworks document, it will open with the development version. You can tell a document is open with the development version by the mode icons (a small gear or pencil) in the upper right corner of each window. This icon lets you switch the

window between user mode to author mode.

User Projects

A Codeworks Project is a collection of classes, objects and scripts that can be used by all Codeworks applications. The standard system classes, such as group and string, are stored in a number of system projects. You can create your own projects that contain useful utility classes and objects that you reuse in a number of different Codeworks applications. These are called *user-projects*.

User projects can also have primitives attached to them. Primitives are scripts that have been written in the language C, and compiled with the PenPoint SDK. Primitives can both provide access to new functionality on your machine, and speed up long computations.

Installing User Projects

If someone else gives you a user project you will need to install it onto your machine. You will need to install to use any Codeworks applications that need it, or if you are going to create an application that uses it.

To Install a User Project:

1. **Drag the user project onto the Settings Notebook on your primary bookshelf.** The system will install the project.

User projects are stored in the Codeworks application section, in the General Software section of the Settings Notebook.

Using a User Project

After you have installed a user project, you can use it any Codeworks application that you create. Each Codeworks application contains a list of user projects that it uses. Initially this list is empty. You can manage this list via the Project Manager.

To open the Project Manager:

1. **Tap on Projects from the Utilities menu (available from the hammer icon).** The projects browser opens.
2. **Tap on Brower-project.** This window changes to display the browser-project.
3. **Tap on Project Manager.** The Project Manager window opens.

   [In the future this will be available from the Utilities menu.]

The Project Manger window lists the user projects that the application uses. You can make the application use a user project with the Load Project button. Remember that a user project must be installed before you can use it.

You can also make the application cease to use a user project with the Unload Project button. This change will only take effect next time you open the application.

Codeworks applications remember which user projects you have loaded and will load them automatically each time they are opened. Remember that every system on which the document is used will have to have the user projects installed.

Creating User Projects

Note: this is an advanced topic. Most Codeworks programmers will not have to know this information. This section assumes your are thoroughly familiar with most of the Codeworks system.

Using the Project Manger window (see User Projects, above) you can create a new user project with the New Project button. You must give the project a name that is unique and distinct from all other user projects. We suggest you use your name or your company name, followed by the purpose of the project. For example: "acme-financial-project" or "bobs-baseball-statistics-project".

Saving User Projects

After you have created a project, the project document will not be created and installed until you tap the Save button in the project browser for the first time. At that point the project will be created and automatically installed (you will see the system installation notes). Unlike applications, projects are not saved automatically: they are only saved when you tap Save. Take extra caution when creating or editing a project: if you turn away from the document without saving, it will lose all your changes.

When you save a project, the system asks you for a comment to describe the changes you are saving. This text is stored in a companion log file with the project. You can use this file (via DOS) to determine the revisions that have been made to a project over time.

Homing Objects in a User Project

To put a class or an object into a project, it must be *homed*. Normally, classes and objects you create have no home and are saved along with the document. System classes, and classes that are in user projects do have a home, which is the project they belong to, and are saved only with that project. This ensures that classes in projects are

shared by all Codeworks documents, rather than each document having its own copy.

To home an object in a user project:

- **Open browser views of both the object to be homed, and the user project to home it in.**
- **Create a local property called 'home' in the object.** A short-cut is to simply drag the home property that the user project has to the object. All user projects have a home.
- **Drag the user project (from its self property) to the value of the newly created home slot in the object.**
- **Tap the Save button in the user project browser window.** You can delay this action till later, but make sure you do it before closing the document, you may lose your objects entirely.

Once an object is homed, it will be saved only with its home project. This means that any changes to such an object must be followed by tapping the Save button on the user project or all changes will be lost.

Primitives and User Projects

[This section discusses how to integrate primitives into a project. It doesn't discuss how to write a primitive.]

When you write a primitive you will be compiling your C language source code with the PenPoint SDK. See the developer files on the Codeworks Development disk for the include (.h) files and sample make and config scripts. You will have to adapt the make scripts to your own version of make (the distribution is for MKS make).

Primitives are always bundled with a user project. You should use the procedures above to create the project. You can distinguish user projects that have primitives because their icons are slightly different: The center hold of the gear is broken to look like a letter 'c'.

When you save the project, the actual project objects are stored in a file called lib.pro and the log file is called

lib.log. You can copy the installed project out of the Settings Notebook on to some other area of your disk, and then look inside of that directory to find these files.

After the first time you create the project, and after each time you Save the project from Codeworks, you must move the lib.pro and lib.log files back to your primitive build directory, or they will be overwritten the next time you build your primitives. The config scripts pick up the lib.pro and lib.log files and move them inside the configured user project.

You may be able to automate these processes by creating make scripts for your version of make that does the appropriate copying of files.

Import & Export

Codeworks applications can import and export files. You write scripts that convert the documents data to and from the external file format (generally ASCII text). By default Codeworks applications neither import nor export. You must set up import and export options, and write their scripts, to enable your applications to import and/or export.

Setting Up Export

To enable an application to export:

1. **Tap on Projects from the Utilities menu (available from the hammer icon).** The projects browser opens.
2. **Tap on penpoint-project.** This window changes to display the penpoint-project.
3. **Tap on Export Options.** The Export Options window opens.
4. **Set Enabled to Yes.**
5. **Fill in the Description field.** This is the name of the export format. This text will be presented to the user when they export the file as one of the export formats. If they choose it, then your export script will be run.
6. **Fill in the File Extension field.** This is a three letter file extension that is used to name the exported file. *[Currently this field is ignored.]*
7. **Tap on Edit Export Script, then write and save the export script.** *[The sample export script is incorrect. The script should take a single argument, file, that is the file object to write on.]*

   [In the future this window will be available from the Utilities menu.]

The export script is passed a file object. The export script should use the write messages to this file object to export the data of the document to the file.

Setting up Import

To enable an application for import:

1. **Tap on Projects from the Utilities menu (available from the hammer icon).** The projects browser opens.
2. **Tap on penpoint-project.** This window changes to display the penpoint-project.
3. **Tap on Import Options.** The Import Options window opens.
4. **Set Enabled to Yes.**
5. **Fill in the Description field.** This is the name of the import type. This text will be presented to the user when they import a file as one of the import options. If they choose it, then your import script will be run.
6. **Choose the Match type you want. For File Extension and File Header matching, also fill in the Match field.** See below for a description of import file matching.
7. **Tap on Edit Import Script, then write and save the import script.** *[The sample import script is blank. The script should take a single argument, file, that is the file object to read from.]*
8. **Close the document and copy it to the settings notebook.** Codeworks applications must be pieces of stationary in the settings notebook in order to import files.

   [In the future this window will be available from the Utilities menu.]

File Matching on Import

When a user imports a file, PenPoint asks each application if the file matches a format it can import. Codeworks applications can match a file in one of four ways:

- **Any File** — matches any file the user imports
- **ASCII File** — matches files that appear to contain only ASCII text. This is done by looking at the beginning of the file. No guarantees are made about the rest of the file.
- **File Extension** — matches any file that has a specified three letter extension at the end of its name.
- **File Header** — matches any file that begins with a given string.

When the user imports a file, if the importing file matches the import options set on a piece of Codeworks stationary, the the user is given the export description from that Codeworks application as one of the import options. If the user chooses that option, then the Codeworks application is copied to the Notebook or Bookshelf (depending on where the user did the import) and then the import script is run.

Stream & File Objects

Streams and Files are objects used for reading and writing text. Streams read and write onto strings. Files read and write onto files used for import and export.

Streams and Files are optimized for text, however, they can also be used to read raw bytes from strings and files.

» The parent of File is Stream. Remember that all of the messages of stream are available to files.

Reading

s **read-byte** *return the numeric value of the next byte*
s **read-char** *return the next character*
These read the next byte from the stream and return either its numeric value in the first case, or the character in the second.

s **read-field** **spacers** w **delimiters** d *return the next field*
This reads the next field from the stream. The text in the stream is read up to and including the next delimiter, which can be any of the characters in the string d. Before the read text is returned, the delimiter, and any white space at either end of the text (defined by characters in the string w) are removed. The arguments spacers and delimiters default to the strings " " and "\t\r\n" respectively. This defaults to reading tab delimited files nicely. If the delimiters includes both CR and LF ("\r\n"), then all three end-of-line standards are handled automatically (see write-line below).

s **read-line** *return the next line*
s **read-word** *return the next word*
s **read-number** **spacers** w **delimiters** d
return the next field converted to a number
These are variants of read-field. The first two default the arguments of field to read lines and words respectively. The third message is just like read-field, only that afterward, the value is converted to a number (using the scan message).

s **read-string** **width** n **delimiters** d *return the next string*
This is similar to read-field, only the field is defined by the width argument, which is the number of characters in the field. If any of the delimiters are reached, then the string is returned, even if it has too few characters. The delimiters argument defaults to "\r\n".

Writing

s **write-byte** n *write a byte with numeric value n*
s **write-char** c *write the character c*
These write a single byte or a character to the stream.

s **write-field** t **delimiter** d *write s followed by d*
s **write-word** t *write s followed by a space*
s **write-number** n **using** f **delimiter** d
format n and then write it followed by d
The first message writes the text t to the stream followed by the delimiter character d. The second does the same, with a delimiter of a space. The last message, first formats the number n using the format f (which default to the normal number formatting), and then writes it followed by the delimiter.

s **write-line** t **dos** *write t followed by a CR-LF sequence*
s **write-line** t **mac** *write t followed by a CR character*
s **write-line** t **unix** *write t followed by a LF character*
s **write-line** t *defaults to dos*
These messages write text out followed by one of the three standards for end-of-line. If the text value t is omitted, then just the end-of-line sequence is written.

s **write-string** t **width** n **max** m *write t out with size constraints*
This writes out t to the stream. If t is shorter than n, then extra spaces are written out to make it n characters long. If t is longer than m, then
only the first m characters are written. Both width and max default to the size of t.

Control

s.position *get the current position*
s.position := n *set the current position*
These get or set the current position in the stream. This is the index of the next character from the stream to read. *[There is a bug in that files are still zero based].*

s.at-end *return true if there are no more characters*

s **skip** n *move the current position by n*
The skip value, n can be negative to move backwards.

s.size *return the size of the stream*
[There is a bug in that this only works for files.]

Creation

stream **over** t *return a new stream, reading the string t*

new file **named** n *open a file, create it if needed*
new file **named** n **create** *delete the file if it exists, the create it*
new file **named** n **exist** *generate an error if the file doesn't exist*

These create a new file object on the file named n. You do not need to use this message for import and export as the file object is passed to your scripts as an argument.

The name determines where the file is stored in PenPoint's file structure, which is only visible to programmers. If the file name is just a simple name (doesn't start with a backslash character) then the file will be created inside a directory called FILES in the document directory. If the file name begins with a single backslash, then it names a path on the current volume. If the file name begins with two backslashes, then it names a volume and a full file path.