PenPoint™ Development Tools
PenPoint™

PenPoint™
Development Tools
PenPoint Application Writing Guide provides a tutorial on writing PenPoint applications, including many coding samples. This is the first book you should read as a beginning PenPoint applications developer.

PenPoint Architectural Reference Volume I presents the concepts of the fundamental PenPoint classes. Read this book when you need to understand the fundamental PenPoint subsystems, such as the class manager, application framework, windows and graphics, and so on.

PenPoint Architectural Reference Volume II presents the concepts of the supplemental PenPoint classes. You should read this book when you need to understand the supplemental PenPoint subsystems, such as the text subsystem, the file system, connectivity, and so on.

PenPoint API Reference Volume I provides a complete reference to the fundamental PenPoint classes, messages, and data structures.

PenPoint API Reference Volume II provides a complete reference to the supplemental PenPoint classes, messages, and data structures.

PenPoint User Interface Design Reference describes the elements of the PenPoint Notebook User Interface, sets standards for using those elements, and describes how PenPoint uses the elements. Read this book before designing your application’s user interface.

PenPoint Development Tools describes the environment for developing, debugging, and testing PenPoint applications. You need this book when you start to implement and test your first PenPoint application.
Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book and Addison-Wesley was aware of a trademark claim, the designations have been printed in initial capital letters.

The authors and publishers have taken care in preparation of this book, but make no expressed or implied warranty of any kind and assume no responsibility for errors or omissions. No liability is assumed for incidental or consequential damages in connection with or arising out of the use of the information or programs contained herein.

Copyright ©1991–92 GO Corporation. All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without prior written permission of the publisher. Printed in the United States of America. Published simultaneously in Canada.

The following are trademarks of GO Corporation: GO, PenPoint, the PenPoint logo, the GO logo, ImagePoint, GOWrite, NoteTaker, TableServer, EDA, MiniNote, and MiniText.

Words are checked against the 77,000 word Proximity/Merriam-Webster Linguibase, ©1983 Merriam Webster. ©1983. All rights reserved, Proximity Technology, Inc. The spelling portion of this product is based on spelling and thesaurus technology from Franklin Electronic publishers. All other products or services mentioned in this document are identified by the trademarks or service marks of their respective companies or organizations.

PenTOPS Copyright © 1990-1992, Sitka Corporation. All Rights Reserved.

GO CORPORATION MAKES NO WARRANTIES, EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT, REGARDING PENPOINT SOFTWARE OR ANYTHING ELSE. GO Corporation does not warrant, guarantee, or make any representations regarding the use or the results of the use of the PenPoint software, other products, or documentation in terms of its correctness, accuracy, reliability, currentness, or otherwise. The entire risk as to the results and performance of the PenPoint software and documentation is assumed by you. The exclusion of implied warranties is not permitted by some states. The above exclusion may not apply to you.

In no event will GO Corporation, its directors, officers, employees, or agents be liable to you for any consequential, incidental, or indirect damages (including damages for loss of business profits, business interruption, loss of business information, cost of procurement of substitute goods or technology, and the like) arising out of the use or inability to use the documentation or defects therein even if GO Corporation has been advised of the possibility of such damages, whether under theory of contract, tort (including negligence), products liability, or otherwise. Because some states do not allow the exclusion or limitation of liability for consequential or incidental damages, the above limitations may not apply to you. GO Corporation’s total liability to you from any cause whatsoever, and regardless of the form of the action (whether in contract, tort [including negligence], product liability or otherwise), will be limited to $50.

The PenPoint documentation is provided with RESTRICTED RIGHTS. Use, duplication, or disclosure by the U.S. Government is subject to restrictions as set forth in FAR 52.227-19 (Commercial Computer Software—Restricted Rights) and DFAR 252.227-7013 (c) (1) (ii) (Rights in Technical Data and Computer Software), as applicable. Manufacturer is GO Corporation, 919 East Hillsdale Boulevard, Suite 400, Foster City, CA 94404.

ISBN 0-201-60861-8
123456789–AL–9695949392
First printing, June 1992
Preface

*PenPoint Development Tools* provides detailed information on the tools and facilities available to developers who are writing programs for the PenPoint™ operating system. This volume includes descriptions of the other volumes in the PenPoint SDK (software development kit), how to run the PenPoint operating system on a desktop PC, how to use the PenPoint source code debugger, and how to use other miscellaneous tools.

This volume also contains a Master Index to the entire SDK documentation set.

Intended Audience

*PenPoint Development Tools* is written for people who are designing and developing applications and other programs for the PenPoint operating system. This book addresses both those who are using high-level development tools (application generators and the like) and those who are writing programs that directly access the PenPoint APIs (application programmatic interfaces). However, parts 2 and 3 of this book will be more useful to those writing programs that access the PenPoint APIs.

What's Here

*PenPoint Development Tools* is divided into several parts:

- **Part 1: Getting Started**, describes what is involved in developing PenPoint programs. It describes the PenPoint documentation and how to run the PenPoint operating system on a desktop PC.

- **Part 2: Debugging PenPoint Programs**, describes the PenPoint source-level debugger, the PenPoint mini-debugger, and other debugging tools and techniques available to developers.

- **Part 3: Tools**, describes other tools that can assist you in developing or embellishing your application. These tools include DOS tools, a bit-map icon editor, a screen shot utility, and a font editor.

Other Sources of Information

The *PenPoint Application Writing Guide* provides a tutorial on writing PenPoint applications. The tutorial is illustrated with several sample applications.

The *PenPoint Architectural Reference* groups the PenPoint classes into several functional areas and describes how to use these classes. The *PenPoint Architectural Reference* is divided into two volumes. The first volume describes the fundamental classes that all application developers will use; the second volume describes supplemental classes that application developers may, or may not, use.
The PenPoint API Reference is a set of "datasheets" that were generated from the PenPoint SDK header files. These datasheets contain information about all the messages defined by the public PenPoint classes. If you own the PenPoint SDK, you can also find the header files in the directory \PENPOINT\SDK\INC.

To learn how to use PenPoint, you should refer to the PenPoint user documentation. The user documentation is included with the PenPoint SDK, and is usually packaged with a PenPoint computer. The user documentation consists of these books:

- *Getting Started*, a primer on how to use PenPoint.
- *Using PenPoint*, a detailed book on how to use PenPoint to perform tasks and procedures.
- *Using GOWrite*, which helps users to develop more effective handwriting when using the GOWrite handwriting translation engine.

**Type Styles in This Book**

To emphasize or distinguish particular words or text, we use different fonts.

**Computerese**

We use fonts to distinguish two different forms of "computerese":

- C Language keywords and preprocessor directives, such as `switch`, `case`, `#define`, `#ifdef`, and so on.
- Functions, macros, class names, message names, constants, variables, and structures defined by PenPoint, such as `DPrintf()`, `msgListAddltem`, `clsList`, `stsBadParam`, `P_LIST_NEW`, and so on.

Although all these PenPoint terms use the same font, you should note that PenPoint has some fixed rules on the capitalization and spelling of messages, functions, constants, and types. By the spelling and capitalization, you can quickly identify the use of a PenPoint term.

- Classes begin with the letters "cls," for example `clsList`.
- Messages begin with the letters "msg," for example `msgNew`.
- Status values begin with the letters "sts," for example `stsOK`.
- Functions are mixed case with an initial upper case letter and are terminated with open and close parenthesis, for example `OSMemAvailable()`.
- Constants are mixed case with an initial lower case letter, for example `wsClipChildren`.
- Structures and types are all upper case (with underscores, when needed, to increase comprehension), for example, U32 or `LIST_NEW_ONLY`.

**Code Listings**

Code listings and user-PC dialogs appear in a fixed-width font.
// Allocate, initialize, and record instance data.
//
StsJmp(OSHeapBlockAlloc(osProcessHeapId, SizeOf(*plnst), &plnst), s, Error);
pInst->placeHolder = -1L;
ObjectWrite(self, ctx, &plnst);

Less significant parts of code listings are grayed out to de-emphasize them. You needn't pay so much attention to these lines, although they are part of the listing.

ObjCallJump(msgNewDefaults, clsAppMgr, &new, s, Error);
new.object.uid = clsTttApp;
new.object.key = 0;
new.cls.pMsg = clsTttAppTable;
new.cls.ancestor = clsApp;
new.cls.size = SizeOf(P_TTT_APP_INST);
new.cls.newArgsSize = SizeOf(APP_NEW);
new.appMgr.flags.stationery = true;
new.appMgr.flags.accessory = false;
strcpy(new.appMgr.company, "GO Corporation");
new.appMgr.copyright = "\213 1992 GO Corporation, All Rights Reserved."
ObjCallJump(msgNew, clsAppMgr, &new, s, Error);

Placeholders

Anything you do not have to type in exactly as printed is generally formatted in italics. This includes C variables, suggested filenames in dialogs, and pseudocode in file listings.

Other Text

The documentation uses italics for emphasis. When a part uses a significant term, it is usually emphasized the first time. If you aren't familiar with the term, you can look it up in the Glossary in the PenPoint Application Writing Guide or the index of the book.

DOS filenames such as \BOOT\PENPOINT\APP are in small capitals. PenPoint file names, which can be upper and lower case, are shown in smaller type, such as \My Disk\Package Design Letter.

Book names such as PenPoint Application Writing Guide are in italics.
Part 1 / Getting Started


**List of Figures**

2-1 Documents in the Software Development Kit 10
3-1 Installing an Application 52

**List of Tables**

2-1 Sample Code in `\PENPOINT\SDK\SAMPLE` 12
2-2 SDK Contents 22
3-1 Tested Machine Configurations 26
3-2 Machine Configurations with No Committed Support 26
3-3 PenPoint Initialization Files in `\PENPOINT\BOOT` 28
3-4 MIL.INI Keywords 33
3-5 ENVIRON.INI Keywords 36
3-6 SYSCOPY.INI Files 43
3-7 UniPen Commands 57
Chapter 1 / Welcome

Welcome to developing applications for the PenPoint™ operating system.

This manual has several aims:

- It tells you what you need to know before starting to develop applications for PenPoint.
- It tells you how the PenPoint SDK documentation is organized.
- It tells you the PC equipment you need to compile and run PenPoint programs.
- It tells you how to run PenPoint on a PC.

Development Options

There are two ways to develop applications on PenPoint:

- Using high-level development tools, such as form builders, 4GL languages, and scripting languages. These tools allow you to write applications without having to access the PenPoint APIs. Using high-level development tools, you can develop applications quickly without needing to learn many of the details of PenPoint. This is at the cost of some speed and flexibility.

- Using a compiled language (such as C) and the PenPoint APIs. This method allows you to write applications that access the PenPoint APIs directly. While the development time is longer, the resulting applications can be faster and more flexible.

Both options have their advantages and disadvantages; your choice of one over the other depends on several factors, including the type of problem you are solving, the existence of a development tool that can provide a solution to this problem, the time available to develop a solution, and your performance requirements.

Using High-Level Development Tools

Almost all high-level development tools run on PenPoint running on either tablet hardware or desktop PCs. If you use a desktop PC, you need:

- A desktop PC that runs PenPoint (see list of supported machines in Chapter 3).

- A digitizer tablet.

- A PenPoint SDK.

- The development tool you will be using to develop your applications.
If you use high-level development tools, you do not need all the extended SDK documentation. Most of the documentation is aimed at developers who are writing PenPoint applications using the PenPoint API.

**Using the PenPoint APIs**

So that you can start to get an idea of how you develop applications for PenPoint, here is what you do to create a PenPoint application:

1. **Design Your Application.** Consider the user interface that you need, how you intend to interact with other PenPoint applications, what devices you need to access, and how you intend to store your data.

2. **Decide what existing classes you can use.** PenPoint provides a rich set of classes that can do much of the work for your application. Your task is to decide which of these classes will serve you best. The task of the *PenPoint Architectural Reference* is to help you find the classes you need.

3. **Decide what new classes you need to create (and their messages).** Perhaps the PenPoint classes don't do exactly what you need. Look for the class that comes closest to your needs, then create your own class that inherits behavior from that class.

4. **Write the application.** PenPoint applications are written in the C language; the object oriented extensions are provided through function calls. Your application uses the PenPoint Application Framework, which performs much of the standard application tasks for you.

5. **Compile, install, test, and debug.** This is the classical cycle. A good strategy is to create an empty application that does no work, but still will appear as an empty page in the Notebook. Then you add the code that creates a new object or defines a new class, compile, install, test, and so on until you have a completed application.

6. **Pretty up the application.** When your application is nearly done, you should create icons, help notebook pages, quick help, stationery, and many other things that make your application "real."

7. **Contact GO Developer Technical Support.** Before you can release a PenPoint application, you need to register your classes with GO. This ensures that no other PenPoint developer will use your class's unique identifiers (UIDs).

8. **Ship product, schedule celebration.** Of course, don’t forget the documentation.

This information is extracted from the *PenPoint Application Writing Guide*, which describes in detail how to create an application. However, before you get there, there are a number of concepts that we need to cover.

Clearly you need to have some familiarity with programming in the C language to write PenPoint applications. In addition to the C language proper, you should also be familiar with DOS C development environments and tools, such as MAKE.
Writing PenPoint application also requires some familiarity with object-oriented programming and techniques, although the *PenPoint Application Writing Guide* spends some time explaining concepts such as classes, objects, messages, inheritance, ancestors and so on.

On occasion, debugging PenPoint programs may require some familiarity with Intel assembly language.

**Additional Reading**

To develop applications that access the PenPoint APIs, you need to know how to program in the C language and you should be familiar with DOS C development environments. If you need further help, there are many good books available on these subjects.

For a description of the 80386 architecture and instruction set (which you need for debugging), see:


Writing PenPoint applications definitely requires familiarity with object-oriented programming. Although the SDK documentation covers this topic to some extent, these books provide much more information on object-oriented programming:


Chapter 2 / The PenPoint Documentation

The PenPoint™ operating system is a rich operating system that makes most of its system-defined classes available to application developers. In fact, you must use some of the classes to create a PenPoint application.

However, with this embarrassment of riches comes some difficulty in knowing exactly what you want or need from the operating system.

This chapter has two purposes. The first is to give you a guide to the PenPoint SDK documentation, its organization, and how we intend you to read it. In describing the documentation, we accomplish the second purpose, which is to give you an overview of the components and features of PenPoint.

If you plan to develop PenPoint applications using a high-level development tool, you may find this tour of the documentation useful. It can help you to decide whether you want to develop using a high-level development tool or the PenPoint APIs.

A Suggested Approach to Documentation

The point of the PenPoint developer documentation is to teach you:

- How to write applications.
- How to find your way around these many classes.

The manuals that make up the SDK documentation library are:

- PenPoint Development Tools (this manual)
- PenPoint Application Writing Guide
- PenPoint UI Design Reference
- PenPoint Architectural Reference (two volumes)
- PenPoint API Reference (two volumes)

The books Getting Started with PenPoint, Using PenPoint, and Using GOWrite teach you how to use the PenPoint operating system. These books are part of the user documentation set that accompanies a pen computer with the PenPoint operating system. They are not considered part of the developer documentation set.
This figure shows the PenPoint SDK documentation library and how we suggest you should use it.

**Feedback on Documentation**

The technical documentation team is eager to get your comments and feedback on the documentation. There is a Reader's Comments Form in the back of each volume of the PenPoint SDK documentation. If you have suggestions, criticisms, or even compliments on the documentation, please let us know. It certainly helps us to know what we're doing right, in addition to what we're doing wrong.

You can fax us your Reader's Comment Form, and any marked-up pages, at (415) 345-9833.
The Application Writing Guide provides a tutorial on how to write PenPoint applications.

System Overview provides an architectural view of the PenPoint operating system. The part describes the kernel, system, component, and application layers of PenPoint, and also touches on the Application Framework and the development environment.

Application Concepts presents applications from a conceptual point of view. This chapter describes most of the parts of an application that you must write, and also describes parts of applications that are provided for you by the PenPoint Application Framework.

A Developer’s Checklist presents a checklist that you can use to ensure that your application is complete. The subsequent sections of the PenPoint Application Writing Guide describe in detail how to implement the parts of the checklist.

Designing Your Application discusses the points that you must consider when creating an application.

Compiling PenPoint Programs describes how to compile and link a PenPoint application.

Running PenPoint describes how to run PenPoint on a PC, and how to install programs in PenPoint.

A Simple Application introduces a minimal application. Through this application, you can see just how much the PenPoint Application Framework does for you.

Debugging describes the tools available to you to test and debug PenPoint Programs.

Creating Objects describes how you create instances of predefined PenPoint classes and use these objects in your application.

Using Windows describes how to use some of the most useful classes in PenPoint, the windows classes. The chapter describes how to create custom windows and presents additional information about using instance data.

Saving and Restoring Data describes how to save and restore data from your application.

Adding Classes describes how you create new classes and add them to PenPoint.

Refining the Application describes how to add polish to your application.

Releasing the Application describes the steps necessary to make your application available to other PenPoint users.
The Appendix describes the programs referred to in the previous chapters and other sample applications in \PENPOINT\SDK\SAMPLE and provides source code listings for some of these applications.

Glossary provides a list of terms used in the PenPoint SDK documentation.

There are more sample applications in the SDK than are described in Application Writing Guide: Sample Code. These sample applications are listed in Table 2-1.

<table>
<thead>
<tr>
<th>Directory</th>
<th>What it Is</th>
<th>What it Demonstrates</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADDER</td>
<td>handwriting calculator</td>
<td>How to write insertion pads and translators for handwritten input.</td>
</tr>
<tr>
<td>BASICSVC</td>
<td>Basic service</td>
<td>The fundamental parts to a PenPoint service.</td>
</tr>
<tr>
<td>CALC</td>
<td>Calculator accessory</td>
<td>Separate &quot;engine&quot; and application objects, table layout, custom layout, labels, filing.</td>
</tr>
<tr>
<td>CLOCK</td>
<td>Clock accessory</td>
<td>Embedding, option sheets, modified application defaults, system timers.</td>
</tr>
<tr>
<td>CNTRAPP</td>
<td>App developed in ADC labs</td>
<td>Saving and restoring application state, memory-mapping state data.</td>
</tr>
<tr>
<td>EMPTYAPP</td>
<td>Minimal application</td>
<td>How the PenPoint application framework performs application boilerplate work.</td>
</tr>
<tr>
<td>HELLO</td>
<td>Graphical “Hello World”</td>
<td>Graphics, text, font scaling, subclassing clsWin.</td>
</tr>
<tr>
<td>HELLOTK</td>
<td>UI Toolkit “Hello World”</td>
<td>Custom layout, labels.</td>
</tr>
<tr>
<td>INPUTAPP</td>
<td>Pen input demo</td>
<td>Pen input, responding to input events.</td>
</tr>
<tr>
<td>MILSVC</td>
<td>PenPoint device driver</td>
<td>How to write a PenPoint MIL service (or device driver).</td>
</tr>
<tr>
<td>NPAPP</td>
<td>Note paper application</td>
<td>Using the NotePaper class that supports ink as a data type.</td>
</tr>
<tr>
<td>PAINT</td>
<td>Painting application</td>
<td>Using pixel maps, drawing contexts, pen input, and non-standard menu bars.</td>
</tr>
<tr>
<td>TEMPLTAP</td>
<td>Template application</td>
<td>A template for starting new applications.</td>
</tr>
<tr>
<td>TESTSVC</td>
<td>A test service</td>
<td>Compiling, installing, and running services.</td>
</tr>
<tr>
<td>TKDEMO</td>
<td>UI Toolkit demo</td>
<td>Using most of the classes in the UI toolkit (borders, labels buttons, fields, menus, custom layouts, table layouts, toolkit tables, notes, option sheets, etc.).</td>
</tr>
<tr>
<td>TTT</td>
<td>Tic-tac-toe application</td>
<td>Gestures, keyboard input, simple graphics, import/export, move/copy, undo, application menus, view-data model.</td>
</tr>
<tr>
<td>WRITERAP</td>
<td>Handwriting recognizer</td>
<td>menus, subclassing clsSPaper, using translator objects, constraining recognition.</td>
</tr>
</tbody>
</table>
**PenPoint Development Tools**

This volume, *PenPoint Development Tools*, provides a description of the PenPoint development environment and documents the tools you use when developing applications.

*Getting Started* describes the overall organization of the SDK documentation and how to run PenPoint on a PC. If you are developing an application using high-level development tools, this is probably all that you need to read.

*Debugging PenPoint Programs* describes the PenPoint debugging tools, including the source-level debugger (DB), the mini-debugger, and the System Log application.

*Development Tools*, describes DOS and PenPoint utilities that can aid you in program development and refinement. Included in this part are DOS tools (such as STAMP, MAKLABEL, and GDIR), the font editor, the on-line tags facility, the PenPoint Bitmap editor (which you use to create icons), and the PenPoint screen shot utility.

*Master Index* contains all the index listings for *PenPoint Application Writing Guide*, *PenPoint Development Tools*, *UI Design Reference*, and the *PenPoint Architectural Reference* volumes. It does not include index entries for the *PenPoint API Reference*.

**PenPoint UI Design Reference**

The *PenPoint UI Design Reference* has two parts:

*PenPoint UI Specification* describes the PenPoint Notebook User Interface (NUI) as it is implemented in PenPoint.

*PenPoint UI Design Guidelines* suggests how you should use the PenPoint NUI when designing your applications.

**PenPoint Architectural Reference**

This is the bible of all the published software interfaces to PenPoint. It is divided into several parts. Each Part talks about some subsystem in PenPoint, listing its features, explaining its concepts, describing its class hierarchy, and discussing how to use it.

The following sections describe the parts in the *PenPoint Architectural Reference* and attempts to summarize the features of the classes that each part documents.

**Class Manager**

*Part 1: Class Manager* describes the PenPoint class manager.

The Class Manager is a subsystem that supports object-oriented programming in PenPoint. The Class Manager provides the message-passing capability in PenPoint, and it provides the mechanisms for object creation and class inheritance. The PenPoint Class Manager and the built-in classes, together with the SDK
Development Tools, comprise a complete object-oriented software development environment for the rapid prototyping and delivery of finished applications in PenPoint.

The PenPoint Class Manager supports these features:

**Message handling.** The Class Manager provides a set of C functions and macros for message handling. These are standard C constructs; there is no special language support for message passing as in C++ or Smalltalk.

**Object Creation.** To create an object, you send the message `msgNew` to a class. The Class Manager does the rest. To create a new class you send the message `msgNew` to a special class, `clsClass`. To copy an object, you send the message `msgCopy` to the object to copy.

**Class Inheritance.** New classes can inherit functionality from ancestor classes.

**Root Classes.** The Class Manager defines two root classes, `clsObject` and `clsClass`, from which all other classes are derived.

**Message types.** Macros are provided for building message tokens. Tokens are differentiated according to the classes that defined them to provide a separate name space for each class.

**Synchronous and asynchronous message passing.** Because PenPoint is a multi-tasking operating system, you send messages to other tasks. If you want to wait for a response from a message that you send to another task, you can use synchronous message passing. If you don't want to wait for a response, you can use asynchronous message passing.

**Single occurrence of executable code.** Executable code that supplies an object’s behavior is not duplicated in the class hierarchy. If the behavior is inherited from an ancestor class, the code for that behavior remains in the ancestor class.

**Observing objects.** You can set up an object so that it has observers. Observers are other objects that are interested in being notified of particular state changes in the observed object. Objects that want to become observers of a target object need only send the target a message requesting observer status.

**Unique Identifiers.** Each object in PenPoint is identified with a unique identifier (UID). These identifiers can be constant (well-known) or dynamic. You use well-known UIDs to identify classes and other objects that must be known to any client that wants to use them. In order to make well-known UIDs unique, a portion of the identifier value is administered by GO Corporation. Contact GO Developer Support to obtain unique UID numbers.
**Application Framework**

*Part 2: Application Framework* describes the PenPoint Application Framework, which provides you the tools you use to allow your application to run under the notebook metaphor. It provides mechanisms for:

- Installing applications on PenPoint (used by the Installer).
- Creating documents.
- Activating documents (turning to a page in the notebook).
- Saving and restoring document data.
- Deactivating and deleting documents (turning away from the page).
- De-installing applications.
- Embedding documents in other documents, which is the foundation for the PenPoint Embedded Document Architecture (EDA).

**Windows and Graphics**

*Part 3: Windows and Graphics* describes ImagePoint, the imaging system for the PenPoint operating system, and how applications can control the screen (or other output devices).

The window system supports:

- Multiple overlapping windows in a window tree.
- Windows on any windowing device; windowing devices include the screen, printers, and memory.
- Many windows. Hundreds can be on-screen at once.
- Control over window clipping, visibility, and transparency.
- Notification when windows are dirty and need to repaint.
- A window layout protocol which lets windows dynamically size themselves and their children.

The ImagePoint imaging model supports:

- Device-independence.
- Arbitrary coordinate system with client-specified:
  - units
  - scaling
  - rotation
  - translation
- RGB color specifications.
- Fill and stroke painting model.
- Polylines, rectangles, ellipses, Bezier curves, arcs, chords.
- Control over line thickness, ends, and corners.
• Gray-scale sampled images ("bitmaps"), including dithering and scaling.
• Outline fonts with these additional features:
  • Hints for rendering fonts in small sizes.
  • Missing fonts are synthesized from closest matching font.
  • Multiple font encodings supported.
  • Substitution for missing characters in the current font.
  • Font editor (supplied as part of the SDK).

**UI Toolkit**

Part 4: *UI Toolkit* describes the PenPoint classes that implement many of the common features required by the PenPoint user interface. These features include:

• Borders, which support some common features of toolkit window repainting, such as margins, and background and foreground colors.

• General layout classes, which implement approaches for positioning and sizing child windows.

• Controls, which implement the translation of window input messages into messages sent to self and other objects.

• Labels, which are a simple way of displaying small amounts of text.

• Buttons, which are labels that the user activates.

• Toolkit tables, which support the initialization, layout and notification management for groups of toolkit components (such as buttons). The capability to organize components in a group is used by several subclasses.

• Menus and menu buttons. Menus are special toolkit tables that often group several buttons and menu buttons; menu buttons are special buttons that display a menu.

• Scroll bars, a special descendant of borders that handles a lot of the work of scrolling for you.

• List boxes, which are scrolling windows that support very large numbers of entries. Unlike a table, only those entries currently visible in the list box need have a window. Descendants of list boxes provide scrolling lists specifically for strings and for font names.

• Text fields, which are labels that the user can write upon. Subclasses of text fields have additional semantics to support integer fields, date fields, fixed-point fields, and so on.

• Notes, which present transient information to the user. PenPoint also provides standard message functions that display information using notes.

• Frames, which manage a collection of other UI components and a client window. Most applications use frames for their main windows, dialog boxes, and pop-up windows. Frames can have decorations, such as close boxes, title bars, tab bars, and command bars.
Option sheets, which present the user with choices for attributes or settings. Because the user can leave an option sheet on-screen, the interaction between option sheets, option cards, and the selection is necessarily quite complex.

Icons, which show an iconic menu button made up of a bitmap image and a string.

Trackers, which grab input and draw transient "rubber-banding" figures in response to pen movements. The toolkit uses this to provide feedback when the user drags or resizes items.

Progress bars, which graphically display the relationship between two values (usually one value represents a total while the second value represents percentage of the total).

**Input and Handwriting**

Part 5: *Input and Handwriting* describes the PenPoint input system and programmatic access to the handwriting translation subsystems. The input system provides:

- Programmatic control of input filtering, including priority of filters.
- Capability to receive all input events not intercepted by the filters (grabbing).
- Routing of input events to the appropriate object (listeners).

The handwriting translation subsystem provides:

- Handling of accumulated pen input (strokes).
- Shape recognition.
- Gesture interpretation.
- Handwriting translation.

**Text Component**

Part 6: *Text Component* describes the PenPoint facilities that allow any application to provide text editing and formatting capabilities to its users. The text component allows:

- Your code to display both plain and fancy text to the user in one or more text data objects.
- The user to interact with the text to modify both the characters and their appearance.
- The user to transfer all or part of the text from one text data object to another (possibly non-text) object, and vice versa.
- Your code to file text data objects.
- Your code to observe and direct the user's interactions with the text.
Embedded objects, which are used to implement insertion pads and signature pads, and can include graphics, spread sheets and other applications in documents.

There is a difference between displaying text through the graphics subsystem and using the Text component. You can use the graphics subsystem to display characters on the screen, but users can't dynamically manipulate the text. Furthermore, the text component includes paragraph and document attributes that define things such as margins and tabs.

**File System**

*Part 7: File System* describes the PenPoint File System. The File System allows you to:

- Create, open, close, and delete files.
- Read and write file data.
- Copy, rename, and move files and directories.
- Seek to a new position within a file or find out the current byte position within a file.
- Modify file and directory attributes.
- Create user-defined attributes for files and directories.

**System Services**

*Part 8: System Services* describes the function calls that applications can use to access kernel functions, such as memory allocation, timer services, process control, and so on.

PenPoint provides basic OS-level services in areas such as task and memory management, as well as run-time language extensions. This part of the *PenPoint Architectural Reference* explains the use of these services. The PenPoint kernel and run-time libraries provide a level of functionality to the applications programmer collectively called *System Services*.

The kernel interface includes task management functions, memory management functions, and inter-task communication functions; these are the low-level PenPoint controls for the Intel multi-tasking environment.

The run-time libraries include functions for string manipulation, character handling, port I/O, buffer manipulation, memory allocation, and time stamp operations. There is a separate library for fixed-point arithmetic functions, which includes both error-checking and fast non-error-checking routines.

Note that there are no file-handling functions in System Services. The PenPoint File System has its own object-oriented API, described in *Part 7: File System* of the *PenPoint Architectural Reference*. The C run-time library implements most of the C standard I/O functions such as `fopen` and `fread` on top of this object-oriented API.
Utility Classes

Part 9: Utility Classes describes a wide variety of classes that save application writers from implementing fundamental things such as list manipulation, busy clock handling, and so on.

- clsList provides a fundamental set of tools for creating and managing a list of 32-bit values. It is no coincidence that UIDs and pointers are also 32-bits long. You can use these objects to store lists of UIDs or pointers to larger structures and you can pass these list objects to other objects.

- clsStream provides the basic messages used to communicate with a stream device. Many other classes descend from clsStream, such as clsFileSystem (the File System) and clsSio (the Serial Port class).

- The browser allows you to create a browser window or a table of contents on screen so that the user can manipulate the files and directories or documents and sections.

- File import and export uses messages from the browser to import files as PenPoint documents and to export PenPoint documents as files.

- The Selection Manager provides an central manager that keeps track of the selection owner. The selection manager notifies observers when the selection changes.

- The Transfer Class provides the messages and functions that implement the PenPoint operating system transfer protocol, which objects can use to exchange data.

- The Quick Help API provides a simple way to provide help to users. When the user makes a question mark gesture on a window, the quick help manager locates the quick help resources associated with that window and displays the resources on screen.

- The Busy Manager allows applications to inform the user when a time-consuming operation is taking place, thereby reassuring the user that the machine is still running.

- The search and replace API provides the protocol and traversal driver to search and replace text strings in embedded objects.

- The Undo Manager enables applications to respond to the Undo command to undo user interface actions.

- clsByteBuf and clsString implement simple data objects which file byte arrays and null-terminated strings.

- clsTable provides a general-purpose table component using a row and column metaphor to implement random and sequential access to data in a file.

- clsNotePaper provides a user interface to the PenPoint operating system's note-taking building block. Using this building block, your application can provide users with a way of managing ink on screen.
Connectivity

Part 10: Connectivity describes the classes that applications can use to access remote devices. The PenPoint remote interface includes:

- A consistent, object-based interface to ports and devices.
- A service architecture that enables users to install drivers and other non-application programs without rebooting or otherwise interrupting work.
- Support for serial and parallel ports.
- Support for fax and data modems.
- Deferred data transfer through an In box and an Out box.
- Access to networks.
- An address book protocol.

Resources

Part 11: Resources describes how to read, write, and create PenPoint resource files. Resources include the following features:

- Ability to store data and object resources in files which can be replaced by the user.
- Agents that can read and write resources that use unique data formats.
- GO provides agents to handle standard data formats.
- Applications can access a set of resources through predefined resource file lists.
- You can define data resources in a C file and compile the file to a resource file.

Installation API

Part 12: Installation API describes PenPoint support for installing applications, services, fonts, dictionaries, handwriting prototypes and so on.

PenPoint provides an installer through which users install and configure applications, fonts, services, dictionaries, and so on. This frees developers from having to write their own installer. The system-provided installer also frees users from having to learn a different installer for each product that they purchase; they just use the PenPoint installer.

The PenPoint Installer uses several system concepts and components:

- All distribution volumes for PenPoint software have the same file organization.
- Dynamic Link Libraries and control files allow products to specify the components that they require. The PenPoint Installer can then load the components only if they are not present in the system already. The Installer can also unload components when they are no longer needed.
• The installation manager class handles installation. If a product requires special treatment not provided by existing installation managers, you can subclass the installation manager class.

• Application monitors exist for each installed application. The application monitors assist in application installation and deinstallation.

Writing PenPoint Services

Part 13: Writing PenPoint Services describes how to write a PenPoint service.

The PenPoint Services Architecture provides a framework for separately installable, non-application DLLs that provide non-application system extensions to applications, components, and other services. The most common use for services is as MIL services (device drivers), but services can also be used for database engines, E-mail backends, and so on.

Services provide these features:

• Targeting other services to form chains of services.

• Delayed binding to targets (the target doesn’t have to exist when the service is created).

• Ownership of services and restricted access to services, based on ownership.

• Notification of connection and disconnection of devices and services.

• A service manager that manages installed services and controls client access to services.

API Reference

The API Reference provides datasheets for each function and message in each subsystem described in the Architectural Reference. These are generated from the header files in \PENPOINT\SDK\INC.

The parts in the PenPoint API Reference are organized identically to the parts in the PenPoint Architectural Reference. However, within each part, the classes are described in no particular order, other than the relative importance to the central topic of the part. If you are looking for a particular class or message, it is a good idea to use the index.

This volume (PenPoint Development Tools) contains a Master Index to all manuals in the PenPoint SDK documentation set.
## Contents of the SDK

Table 2-2 lists the directories and key files in the PenPoint SDK and describes their contents.

<table>
<thead>
<tr>
<th>Directory or File</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>\PENPOINT\APP</td>
<td>PenPoint applications.</td>
</tr>
<tr>
<td>\PENPOINT\BOOT</td>
<td>Control files, start-up code.</td>
</tr>
<tr>
<td>\PENPOINT\BOOT\APP</td>
<td>PenPoint system applications (Notebook, Bookshelf, etc.).</td>
</tr>
<tr>
<td>\PENPOINT\BOOT_APP</td>
<td>Optional DEBUG versions of system apps.</td>
</tr>
<tr>
<td>\PENPOINT\BOOT\DLL</td>
<td>System DLLs.</td>
</tr>
<tr>
<td>\PENPOINT\BOOT_DLL</td>
<td>Optional DEBUG versions of system DLLs.</td>
</tr>
<tr>
<td>\PENPOINT\BOOT*.INI</td>
<td>Control files.</td>
</tr>
<tr>
<td>\PENPOINT\BOOT*.DLC</td>
<td>Control files of DLLs loaded at start-up.</td>
</tr>
<tr>
<td>\PENPOINT\BOOT_<em>.</em></td>
<td>Backup copies of control files.</td>
</tr>
<tr>
<td>\PENPOINT\BOOT\MIL.OS</td>
<td>Machine Interface Layer code.</td>
</tr>
<tr>
<td>\PENPOINT\BOOT\PENPOINT.OS</td>
<td>PenPoint kernel.</td>
</tr>
<tr>
<td>\PENPOINT\BOOT_IP.OS</td>
<td>DEBUG version of PenPoint kernel.</td>
</tr>
<tr>
<td>\PENPOINT\BOOT\PPBOOT.EXE</td>
<td>DOS program that loads PenPoint.</td>
</tr>
<tr>
<td>\PENPOINT\FONT</td>
<td>PenPoint fonts.</td>
</tr>
<tr>
<td>\PENPOINT\HWX</td>
<td>Handwriting prototypes.</td>
</tr>
<tr>
<td>\PENPOINT\PDICT</td>
<td>Home of personal dictionary.</td>
</tr>
<tr>
<td>\PENPOINT\PREFS</td>
<td>Home for preference settings.</td>
</tr>
<tr>
<td>\PENPOINT\SERVICE</td>
<td>System services used by PenPoint.</td>
</tr>
<tr>
<td>\PENPOINT_SERVICE</td>
<td>Optional DEBUG versions of system services.</td>
</tr>
<tr>
<td>\PENPOINT\SDK</td>
<td>PenPoint applications that are not loaded in the default PenPoint configuration; some of these are SDK tools such as the bitmap editor and S-Shot tool, others are applications that you must distribute with your own application if it requires them.</td>
</tr>
<tr>
<td>\PENPOINT\SDK\APP</td>
<td>Optional DEBUG versions of non-system apps.</td>
</tr>
<tr>
<td>\PENPOINT\SDK\DLL</td>
<td>PenPoint DLLs that are not loaded in the default PenPoint configuration that you must distribute with your own application if it requires them.</td>
</tr>
<tr>
<td>\PENPOINT\SDK\INC</td>
<td>PenPoint header files.</td>
</tr>
<tr>
<td>\PENPOINT\SDK\INC\SYS</td>
<td>Some header files are in this subdirectory for ANSI C compatibility.</td>
</tr>
<tr>
<td>\PENPOINT\SDK\LIB</td>
<td>Library information to enable you to link your code with routines and externals in PenPoint DLLs and PENPOINT.OS.</td>
</tr>
<tr>
<td>\PENPOINT\SDK\SAMPLE</td>
<td>Sample application code and sample Watcom WMAKE Makefiles.</td>
</tr>
<tr>
<td>\PENPOINT\SDK\UTIL\CLSMGR</td>
<td>PenPoint Method Table compiler.</td>
</tr>
<tr>
<td>\PENPOINT\SDK\UTIL\DOS</td>
<td>DOS utilities for PenPoint.</td>
</tr>
</tbody>
</table>

continued
The files on the Goodies disk are unsupported, however you might find them useful. Most software on the Goodies disk has a README.TXT file in its subdirectory that explains its use. To reiterate: software on the Goodies disk is unsupported. PenPoint Developer Support will not answer questions related to anything on the Goodies disk.
Chapter 3 / Running PenPoint on a PC

Although the PenPoint™ operating system is targeted for mobile pen-based computers, it is possible to run PenPoint on some PC configurations. The simulation is imperfect (no static RAM, no pen-on-screen interaction, and so on) but it is useful for development and debugging. The mouse simulation of a pen does not have the same user interface as a PenPoint computer at all. Using a pen tablet attached to a PC lets you hand write more naturally, but you still aren’t able to directly touch objects on-screen or write on what you see.

PenPoint will not work on all PCs, even those advertised as “100% IBM PC compatible,” even those claiming compatibility with the requirements below.

Hardware

To run PenPoint on a PC, you need a 386-class machine with a hard disk that has at least 25 Megabytes free.

If you want to run PenPoint on your PC, in addition it must have the following:

• 8 MB of RAM configurable as extended memory. You may be able to develop with less memory, however the PenPoint Source-level Debugger with full symbol tables and all applications loaded takes around 6 MB.

• IBM VGA or Compaq VGA video adapter (because PenPoint has its own video drivers, adapters advertised as being 100% compatible with these under DOS may not work).

• Either a mouse or a digitizing tablet.

To repeat, your PC may meet these requirements yet not be able to run PenPoint. It may even be able to run one release, yet not another. Table 3-1 lists the computers that GO has tested and supports for PenPoint development.
Table 3-1
Tested Machine Configurations

<table>
<thead>
<tr>
<th>Machine</th>
<th>Configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compaq 386/20E</td>
<td>20MHz 386, no coprocessor, Compaq BIOS dated 31-May-89, DOS 5.0, 640KB + 8192 Extended = 8,832KB total memory.</td>
</tr>
<tr>
<td>Dell 325P</td>
<td>25MHz 386, no coprocessor, Phoenix BIOS dated 27-Sep-91 version 1.10A16, DOS 5.0, 640KB + 7,160 extended = 7,800KB total memory.</td>
</tr>
<tr>
<td>IBM PS2/70</td>
<td>25MHz 386, no coprocessor, IBM BIOS dated 2-Feb-89, DOS 5.0, 640KB + 7,424 = 8,064KB total memory.</td>
</tr>
<tr>
<td>IBM PS2/70</td>
<td>IBM BIOS dated 11-Apr-88, 640KB + 5,376 = 5,980KB total memory.</td>
</tr>
<tr>
<td>IBM PS2/80</td>
<td></td>
</tr>
<tr>
<td>IBM PS2/90</td>
<td>33MHz 486, built-in coprocessor, IBM BIOS dated 30-Jan-91, DOS 5.0, 640KB + 11,520 extended = 12,160KB total memory.</td>
</tr>
<tr>
<td>IBM PS2/95</td>
<td>25MHz 486, built-in coprocessor, IBM BIOS dated 8-Jan-90, DOS 5.0, 640KB + 7,424 extended = 8,064KB total memory.</td>
</tr>
<tr>
<td>NCR 386sx/MC20</td>
<td>20MHz 386, no coprocessor, NCR BIOS version 1.01.00 and Phoenix BIOS dated 15-Oct-90 version 1.02.07, DOS 3.30, 640KB + 7,424 extended = 8,064KB total memory.</td>
</tr>
</tbody>
</table>

GO has also tested PenPoint on the machines listed in Table 3-2, but makes no commitment for continued support.

Table 3-2
Machine Configurations with No Committed Support

<table>
<thead>
<tr>
<th>Machine</th>
<th>Configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Northgate 386 33E</td>
<td>Model FCH, 33MHz 386, no coprocessor, Northgate (AMI) BIOS dated 15-Sep-89 version 3.2B, DOS 4.01, Video 7 VGA version 1.09, 40MB type 22 drive, 640 + 7,552 extended = 8,192KB total memory.</td>
</tr>
<tr>
<td>Samsung S800</td>
<td>20 MHz 386, no coprocessor, AMI BIOS version 3.04 03B dated 30-Nov-87, DOS 5.0, ATI Basic 16 VGA, type 34 Western Digital IDE controller, Conner 201 MB drive.</td>
</tr>
<tr>
<td>Toshiba 5200/100</td>
<td>20 MHz 386, no coprocessor, Phoenix BIOS dated 15-Jan-88, DOS 4.01, built-in VGA, 640 + 7,168 extended = 7,808KB total memory, 102MB drive.</td>
</tr>
<tr>
<td>Uniq 486/33</td>
<td>33 MHz 486, built-in coprocessor, American Megatrends, Inc. (AMI) 486 BIOS for Vantage 486 version 2.0 dated 7-Jul-91, DOS 5.0, ATI VGA Wonder XL VGA w/ 1MB, 640 + 7,424 extended = 8,064KB total memory, type 47 controller.</td>
</tr>
</tbody>
</table>

**Mouse**

If you do not have a digitizing tablet, you can use a Logitech C7 or C9 serial mouse, a Microsoft compatible mouse, a PS/2 mouse, or a bus mouse. GO does not currently support the Logitech Mouseman or newer Logitech mice.

You may need to load the DOS driver for your mouse before starting PenPoint; either specify the device driver MOUSE.SYS in your CONFIG.SYS file or make sure to run MOUSE.COM before you start PenPoint.
Memory, Caches, and RAM Disks

You need 8MB of RAM to run PenPoint. It must be extended memory, not expanded memory. Memory-resident software may use up much of your memory. PenPoint might run in less memory, but our QA process has only evaluated 8 MB machines.

The Watcom compiler and linker (WCC386P and WLINKP) are protected-mode programs that use extended memory. If you use large disk caches or RAM drives, they may not have enough memory to run. Also, the DOS 5 EMM386.EXE driver conflicts with their use of memory. For more information, see the section on compiling.

PenPoint will not work if you run a disk cache such as SmartDrive. It also interferes with some RAM disks. We believe that VDISK can coexist with PenPoint, but others may not.

You can check whether your RAM disk is compatible through these steps:

- Run PenPoint (PPBOOT.EXE).
- Exit back to DOS.
- If the size or contents of your RAM disk has been altered, assume that it and PenPoint conflict.

Networks

To stop running PenPoint on a PC, you either tap the shutdown button in the Preferences Power option sheet or press [Pause] and enter q at the prompt. Usually your computer returns to DOS. However, if your computer has loaded network software, you may have to reboot your computer after running PenPoint (because the machine will hang). If you want to avoid this reboot, do not load your network software before you run PenPoint or remove it from memory. In either case, GO recommends that you run CHKDSK to verify the state of the DOS file system.

Labeling Volumes

Unlike the DOS convention of listing disks by their drive letter (C: for example), PenPoint refers to disks by their volume names (not by their devices). You can name volumes when you format them, or with the DOS LABEL command. It's a good idea to give volume names to all the floppies and hard drives you will be using with PenPoint.

PenPoint volumes are indicated by a \ before the volume name. Thus \MYDISK\DIR\FILE1 is a path on a volume labelled MYDISK.

There are a few reserved volume names in PenPoint. \RAM is reserved for the PenPoint memory file system, which is usually in static RAM, but can be on-disk. \BOOT is the volume from which PenPoint booted (programmatically known as theBootVolume). When running PenPoint on a PC, it's safe to refer to files

Warning You can use PenPoint to create a disk label in lower case letters. Such a label cannot be changed by LABEL in DOS 5.0.
relative to \BOOT; however, when booting a PenPoint computer from floppies, your configuration information must refer to the actual floppy disk volume labels. If PenPoint encounters a volume name in a path that it does not recognize, it will prompt the user to insert the volume with that name.

**Setting Up Initialization Files**

When PenPoint starts, it reads several initialization files in \PENPOINT\BOOT. These set certain environment variables and tell it what dynamic link libraries (DLLs) to load, and what applications to install. Before running PenPoint for the first time, you will probably need to modify at least one of these files.

All files you need to run PenPoint on a PC are in the \PENPOINT directory that you installed with your SDK. The document on installing PenPoint shipped with your software contains the latest information on installing PenPoint and machine requirements to run it.

Default versions of the initialization files necessary to run PenPoint on a PC are in \PENPOINT\BOOT. The SDK contains two versions of each initialization file. One is preceded by an underscore (_), the other isn’t. You should modify the file that does not contain the underscore, and leave the one with the underscore so that you can refer to it later.

If, after changing the originals, you have difficulty running PenPoint, compare the modified file with its underscored equivalent.

Two notes on initialization files:

- The last character in your initialization files must be a newline (carriage return, line feed). If the newline is not present, PenPoint will ignore the last line in your file.
- If the line is of the form tag=value and you have two lines that define the same tag, PenPoint uses the first occurrence and ignores all subsequent occurrences of the tag.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIL.INI</td>
<td>Hardware interface settings for PC MIL.</td>
</tr>
<tr>
<td>ENVIRON.INI</td>
<td>PenPoint configuration settings.</td>
</tr>
<tr>
<td>BOOT.DLC</td>
<td>List of system DLLs required to run PenPoint.</td>
</tr>
<tr>
<td>CONSOLE.DLC</td>
<td>A subset of BOOT.DLC for single-screen debugging.</td>
</tr>
<tr>
<td>SYSCOPY.INI</td>
<td>List of files and directories to copy to memory.</td>
</tr>
<tr>
<td>SYSAIP.INI</td>
<td>System applications to install.</td>
</tr>
<tr>
<td>APP.INI</td>
<td>Other applications to install.</td>
</tr>
</tbody>
</table>
**PenPoint Boot Sequence**

When you run PPBOOT.EXE on a PC, it locates the MILINI file and uses the information in MILINI to configure the hardware, initialize the MIL (machine interface layer), and start the PenPoint operating system.

PPBOOT determines the volume from which PenPoint is being booted, and assigns the boot volume to the symbol `theBootVolume`. You can use `theBootVolume` in your code to refer to this volume.

The operating system locates the ENVIRON.INI file and uses that information to configure things such as the size of the display, the selected volume, the time zone, and the name of the first application to run. If a `\PENPOINT` directory does not exist on the volume, PenPoint creates one.

The PenPoint operating system then looks for `\PENPOINT\BOOT\BOOT.OLC` and it initializes each DLL listed. PenPoint then copies all the files and directories listed in `\PENPOINT\BOOT\SYSCOPY.INI` to the selected volume.

PenPoint installs the system applications listed in `\PENPOINT\BOOT\SYSAPP.INI`. The last DLL in `BOOT.DLC`, AUXNBMGR.DLL, installs the applications listed in `\PENPOINT\BOOT\APP.INI`.

**MILINI**

The file `\PENPOINT\BOOT\MILINI` describes configuration of the PC on which the PenPoint operating system will run. Although most devices are installable, there are a handful of devices that PenPoint must know about ahead of time, such as the screen type, the stylus device, and the system clock.

MILINI is specific to the implementation of the PenPoint MIL (machine interface layer) for a PC. Because PC configurations can vary so much, the PC MIL implementation reads the configuration information from MILINI. The MILs that run on most tablet hardware will be preconfigured for a specific machine and thus will not need to read configuration information from a file.

You must modify `\PENPOINT\BOOT\MILINI` before you can boot. At a minimum, you need to specify your pointing device and monitor in MILINI. Another common change is to specify the port (if any) on which debug output should appear. MILINI contains instructions for how to modify itself together with commented-out versions of common settings.

Changing MILINI does not change an active system. Changes take effect the next time you boot PenPoint.

The format is of a setting in MILINI is:

```
keyword = value
```

Following is a list of the MILINI file shipped with the SDK:

**Important** MILINI is specific to the implementation of the PC MIL. Other MILs probably won't have a MILINI file.
The following applies to all serial pointing devices:

Use the UNIPENPORT tag to set the serial port. For example:
- UNIPENPORT = 1  - This will set unipen to COM1:
- UNIPENPORT = 760,5  - This will set unipen to comport at 2F8 irq 5

Use the UNIPENTYPE tag to select a predefined protocol:
- UNIPENTYPE = WACOM510C  - This sets unipen to the Wacom 510C digitizer

# MICROSOFT: Microsoft serial, two-button mouse
# LOGITECH: Logitech C7 or C9 serial, three button mice. For a Logitech
# MouseMan use the MICROSOFT tag
# WACOM510C: The switches must be set as follows (X = ON, 0 = OFF) -
# DS1 DS2 DS3
# Front OXOXXXOX XXOXOXO XXOXOXO Back
# WACOM510: The older Wacom units, red power LED, attached power cord.
# The switches must be set thusly (X = ON, 0 = OFF) -
# Front OOOXXOXO XXOXOXO XXOXOXO OXOXOXOXO Back
# SuperScriptII: The SuperScript II LCD/digitizer combo. For the ink to
# be aligned you must specify "ScreenType=SuperScript" or
# "ScreenPixelsPerMeter=3690" in MIL.INI.
# CalCompDBII: CalComp’s DrawingBoard II, should also work with their
# “Wiz” product. Use the default “Hi Resolution Binary”
# format (#23) in “run” mode, 9600 baud no parity, eight
# data bits, one stop bit, 1000 lpi resolution, 125 pps.
# Enable “Send when out of proximity”, button 10=1 in bank B.

The following are generic descriptions. You will probably have to “tune”
these by placing the parameters into MIL.INI and adjusting for
the specific characteristics of your digitizer.

# GAZELLE: For products from Gazelle System (now owned by Logitech)
# MM: The common “MM” digitizer protocol.
# BITPAD2: BitPad 2 binary protocol
# BITPAD2ASC: BitPad 2 ASCII protocol

See the universal pen driver manual for details on how to define a
custom serial protocol.

Remove the "#" on the line below for the port of your pointing device:
- UNIPENPORT = 1
- UNIPENPORT = 2

Remove the "#" on the line below which matches your serial pointing device:
- UNIPENTYPE = MICROSOFT
- UNIPENTYPE = LOGITECH
- UNIPENTYPE = CalCompDBII
- UNIPENTYPE = GAZELLE
- UNIPENTYPE = SuperScriptII
- UNIPENTYPE = WACOM510C
- UNIPENTYPE = WACOM510
- UNIPENTYPE = MM
- UNIPENTYPE = BITPAD2
- UNIPENTYPE = BITPAD2ASC
# This supports a Microsoft "bus" or "InPort" mouse.
# Remove the "#" on the next line if your bus mouse is the primary one
#BusMousePort=Primary
# Remove the "#" on the next line if your bus mouse is the secondary one
#BusMousePort=Secondary
# Remove the "#" on the next line to enable the PS/2 mouse port.
# Note: do not confuse a PS/2 mouse port with a Microsoft bus mouse port.
#PS2Mouse=on

# Remove the "#" from "LeftyMouse" if you are left-handed and have a
# two button mouse. This will swap the meaning of the buttons so that
# the right button will mean "tip-down" and the left button "out of prox".
#LeftyMouse=true

# The following lines are concerned with the Wacom "310" coprocessor. They
# can be used to describe the alignment and rotation of the pen sensor
# relative to the LCD panel. The X axis is the long axis, the Y the short.
# All numbers are in decimal (base 10). The defaults for the GO "Hyde"
# prototype are listed:
# Remove the "#" to enable the Wacom 310 MIL device:
#Wacom310=on
# This tag is used to set the address of the command I/O port
#Wacom310Cmd=1222
# This tag is used to set the address of the data I/O port
#Wacom310Data=1220
# This tag is used to set the Wacom 310 interrupt level
#Wacom310Int=10
# These two tags can be used to align the pen in the case of a constant error.
# Signed values are supported. The units are 0.1 millimeters
#Wacom310XOffset=0
#Wacom310YOffset=0
# These two tags describe the maximum values for the X and Y axis
#Wacom310MaxXPos=2240
#Wacom310MaxYPos=1400
# These two tags may be use to independently flip either axis. When used
# together the pen coordinates can be rotated 180 degrees.
#Wacom310InvertX=yes
#Wacom310InvertY=yes
# Remove the "#" on the next line to route debugging information to COM1
#SerialDebugPort=1
# Remove the "#" on the next line to route debugging information to COM2
#SerialDebugPort=2
# Remove the "#" from "MonoDebug" ONLY if you don't want debugging info
# to be sent to an existing monochrome video card (the "second head")
#MonoDebug=off
# Remove the "#" on the line below which matches where your wish to route
# the lowlevel debugging information. If no "LowLevelDebug" is specified then
# these low-level messages will be supressed
#LowLevelDebug=mono
#LowLevelDebug=com1
#LowLevelDebug=com2
# Remove the "#" from "Floppy=hardware" if you wish to enable the PC floppy
# driver
#Floppy=hardware
# Change "off" to "on" in this line to enable auto-polling of floppy drives
# when "floppy=hardware"
#FloppyAttachment=off
# Remove the "#" from "Harddisk=hardware" if you wish to enable the PC hard
# disk driver which talks directly to the hardware
#HardDisk=hardware
# Set Com1 to true to enable Com1 Asynchronous SIO, false otherwise
Com1=false
# Set Com2 to true to enable Com2 Asynchronous SIO, false otherwise
Com2=false
# Set Lpt1 to false to disable Lpt1 high speed packet parallel port I/O
Lpt1=true
# Set Lpt2 to false to disable Lpt2 high speed packet parallel port I/O
Lpt2=true
# Remove the "#" on the line below which matches your screen and video board
# If no "ScreenType" is specified then the MIL defaults to STD480 (a standard
# VGA screen in landscape). An automatic "int10 18" is done by the MIL for
# STD480 to set the VGA card into mode 18 (12 hex): 640 by 480 graphics.
ScreenType=STD480
#ScreenType=EGANul
#ScreenType=NCR
#ScreenType=IBM
#ScreenType=Samsung
#ScreenType=Ymh400
#ScreenType=Hyde
#ScreenType=SuperScriptII
#ScreenType=ATI768

# Each of the above screen types defines a preferred initial orientation. If
# you wish to override this then read on:
#
# Initial screen orientation: for a conventional VGA screen these are:
#
# North
#  -
#  |
#  |
#  |
#  |
#  |
#  |
#  |
# West   |   East
#       |
# (top   |   (bottom
# in     |   in
# char.  |   character
# mode)  |   mode)
#       |
#       |
#       |
#       |
#       |
# ^ South ^
# ^      ^
# ^ last VGA scan line
# ^ first VGA scan line
# Remove the "#" on the line below which matches your desired orientation:
#
#InitialScreenTop=North
#InitialScreenTop=South
#InitialScreenTop=East
#InitialScreenTop=West

# If you have a VGA display on your computer but the machine is not listed
# above then you may explicitly set the pixel spacing by completing the
# line below and removing the "#". Typical values range from 2500 to 3700:
#ScreenPixelsPerMeter=????
# Remove the "#" on the line below which matches the your flash card type.
FLASHCARD=TOSHIBA
FLASHCARD=PC

# Remove the "#" on the line below if you are having trouble exiting
# from PenPoint back to DOS
ExitToDOS=reset

Capitalization in the settings isn’t significant. This table lists the MIL.INI keywords used by PenPoint.

<table>
<thead>
<tr>
<th>Table 3-4</th>
<th>MIL.INI Keywords</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Keyword</strong></td>
<td><strong>Meaning</strong></td>
</tr>
<tr>
<td>UniPenPort</td>
<td>Sets the serial port for universal pen device.</td>
</tr>
<tr>
<td>UniPenType</td>
<td>Sets a predefined universal pen protocol. See the complete discussion of the universal pen driver later in this chapter for details on how to define a custom serial protocol.</td>
</tr>
<tr>
<td>BusMousePort</td>
<td>Specifies whether a bus mouse is secondary to another mouse device.</td>
</tr>
<tr>
<td>PS2Mouse</td>
<td>Enables the PS/2 mouse port.</td>
</tr>
<tr>
<td>LeftyMouse</td>
<td>Switches buttons on two or three button mouse for left-handed users.</td>
</tr>
<tr>
<td>Wacom310</td>
<td>Enables a Wacom 310 coprocessor.</td>
</tr>
<tr>
<td>Wacom310Cmd</td>
<td>Sets the address of the command I/O port.</td>
</tr>
<tr>
<td>Wacom310Data</td>
<td>Sets the address of the data I/O port.</td>
</tr>
<tr>
<td>Wacom310Int</td>
<td>Sets the Wacom 310 interrupt level.</td>
</tr>
<tr>
<td>Wacom310XOffset</td>
<td>Sets a constant X offset.</td>
</tr>
<tr>
<td>Wacom310YOffset</td>
<td>Sets a constant Y offset.</td>
</tr>
<tr>
<td>Wacom310MaxXPos</td>
<td>Specifies the maximum X position.</td>
</tr>
<tr>
<td>Wacom310MaxYPos</td>
<td>Specifies the maximum Y position.</td>
</tr>
<tr>
<td>Wacom310InvertX</td>
<td>Inverts the X axis.</td>
</tr>
<tr>
<td>Wacom310InvertY</td>
<td>Inverts the Y axis.</td>
</tr>
<tr>
<td>SerialDebugPort</td>
<td>Specifies a serial port on which to write debugging information.</td>
</tr>
<tr>
<td>MonoDebug</td>
<td>Turns off debugging information for a second (monochrome) monitor (when off).</td>
</tr>
<tr>
<td>LowLevelDebug</td>
<td>Specifies destination for MIL debugging information.</td>
</tr>
<tr>
<td>Floppy</td>
<td>Enables floppy disks through hardware or BIOS.</td>
</tr>
<tr>
<td>FloppyAttachment</td>
<td>Enables autopolling of the floppy drive to detect attachment.</td>
</tr>
<tr>
<td>HardDisk</td>
<td>Enables hard disks accessed through hardware or BIOS.</td>
</tr>
<tr>
<td>Com1</td>
<td>Enables COM1 asynchronous serial I/O.</td>
</tr>
<tr>
<td>Com2</td>
<td>Enables COM2 asynchronous serial I/O.</td>
</tr>
</tbody>
</table>

**continued**
### ENVIRON.INI File

The file `\PENPOINT\BOOT\ENVIRON.INI` contains a set of environment settings for PenPoint. PenPoint system software and other programs defines names and values in `ENVIRON.INI`. Some of the DLLs use this to get initialization information such as the type of display, the monitor orientation, the default volume, and so on.

You should modify `\PENPOINT\BOOT\ENVIRON.INI` to enable logging by uncommenting the line

```
#DebugSet=/D*l /DD8000
```

This sets up logging to the file `\PENPOINT\LOG` on the boot volume.

This is probably the only change you need to make to the default `ENVIRON.INI` in order to run PenPoint; the default settings for the variables described below are correct.
Like MIL.INI, changing ENVIRON.INI does not change an active system. Changes take effect the next time you boot PenPoint.

The format is of a setting in ENVIRON.INI is:

\[ \text{keyword} = \text{value} \]

This is a list of the _ENVIRON.INI file shipped with the SDK.

```ini
## PenPointPath=
##
## Initial Penpoint application to run
StartApp = \\boot\penpoint\boot\app\bookshelf

## Alternate screen metrics, in device units
ScreenWidth=200
ScreenHeight=320

## Size of the swap file.
## This is number of bytes in swap file, in hexadecimal; e.g. 5MB = 500000.
SwapFileSize=500000

## Boot configuration. Choices are: DebugRAM, DebugTablet and Tablet.
Config=DebugRAM

## Timezone setting used by ANSI time routines (see WATCOM library ref)
TZ=PST8PDT

## Initial setting of the debug flags
## /D*1: Validates all heap allocates and frees under the debug penpoint.os.
DebugSet=/D*1
#DebugSet=/DD8000 /D*1

## Forces a flush after N or more characters are written, 1 flush per call to DebugBuf (called by Debugf, DPrintf).
#DebugLogFlushCount=0

## Version string used by Preferences
Version=PenPoint|386 (386.45H)|Copyright c 1992, GO Corporation|All Rights Reserved.

## Path to a default bookshelf, copied at boot time
#BkshelfPath=\boot\penpoint\boot\doc

## Progress Bar Maximum
bootProgressMax=250

## Open the notebook at boot time
Autozoom=Notebook

## Root of PenPoint
```

Capitalization in the settings isn't significant. The following table lists the ENVIRON.INI keywords used by PenPoint.
### ENVIRON.INI Keywords

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>AutoZoom</td>
<td>Specifies the name of a document to open in the window after booting.</td>
</tr>
<tr>
<td>BkShelfPath</td>
<td>Specifies the path to the bookshelf directory.</td>
</tr>
<tr>
<td>BootProgressMax</td>
<td>Used in the booting progress indicator. Do not modify.</td>
</tr>
<tr>
<td>Config</td>
<td>Specifies the disk and debugging environment.</td>
</tr>
<tr>
<td>DebugLog</td>
<td>Specifies the name of a file for debugger output.</td>
</tr>
<tr>
<td>DebugLogFlushCount</td>
<td>Specifies the when to flush the debug log to a file.</td>
</tr>
<tr>
<td>DebugSet</td>
<td>Specifies one or more debug flags.</td>
</tr>
<tr>
<td>PenPointPath</td>
<td>Specifies the location of the PENPOINT directory.</td>
</tr>
<tr>
<td>PenProxTimeout</td>
<td>Specifies a delay before sending an out of proximity event.</td>
</tr>
<tr>
<td>ScreenHeight</td>
<td>Specifies the height of the screen in device units.</td>
</tr>
<tr>
<td>ScreenWidth</td>
<td>Specifies the width of the screen in device units.</td>
</tr>
<tr>
<td>StartApp</td>
<td>Specifies the first PenPoint application to run on startup.</td>
</tr>
<tr>
<td>StealMem</td>
<td>Specifies that PenPoint should use less memory.</td>
</tr>
<tr>
<td>SwapBoot</td>
<td>Specifies that PenPoint should load memory from a saved swap file.</td>
</tr>
<tr>
<td>SwapFileSize</td>
<td>Specifies the size of the swap file in bytes.</td>
</tr>
<tr>
<td>TZ</td>
<td>Specifies the time zone.</td>
</tr>
<tr>
<td>UndoLimit</td>
<td>Specifies the maximum number of undos.</td>
</tr>
<tr>
<td>Version</td>
<td>Specifies the PenPoint version label.</td>
</tr>
<tr>
<td>VolSel</td>
<td>Specifies the label of the volume used to create the PenPoint system.</td>
</tr>
<tr>
<td>WinMode</td>
<td>Specifies the initial orientation of the screen device.</td>
</tr>
<tr>
<td>ZoomMargin</td>
<td>Specifies the distance from the bottom of the autozoom document to the bottom of the screen.</td>
</tr>
<tr>
<td>ZoomResize</td>
<td>Specifies whether the autozoom document has a resize tab.</td>
</tr>
</tbody>
</table>

An application can add its own keywords to ENVIRON.INI for testing. To get a value from ENVIRON.INI, use the kernel function `OSEnvSearch()`.

These sections describe the keywords in more detail:

#### AutoZoom

AutoZoom specifies the name of a document that PenPoint is to open on the desktop after booting. Usually this document is Notebook (meaning the main note book). However, you can specify any other document, provided its application is installed.

If you set ZoomMargin to 0 and specify ZoomResize as `FALSE`, the document that is opened with AutoZoom is locked into the page. If the document is a notebook, the user will not be able to close or resize the notebook. If the document is a page-based application, the user will not be able to turn away from the application.
BkShelfPath

BkShelfPath specifies the path to the bookshelf directory. Usually this contains \PENPOINT\BOOT\DOC.

BootProgressMax

BootProgressMax is used by the progress indicator that appears when you boot PenPoint. Do not modify this field.

Config

Config is used to specify in one place the selected volume, the location of installed code, and, in the future, debugging options. There are three environments that you can specify with Config:

- **DebugRAM** Sets theSelectedVolume to RAM and stores installed code in RAM also. In this configuration, any changes you make when running PenPoint are not saved, but it is easier to clean up after and it makes it easy to install new versions of applications after each reboot.

- **Tablet** Sets theSelectedVolume to be the drive on your PC from which you booted PenPoint and stores installed code on the disk. In this configuration, changes that you make are saved from one cold boot to the next. For other considerations on Config=Tablet, see the section Running in Tablet-Like Mode.

- **DebugTablet** Currently is similar to Tablet. In the future, DebugTablet may be expanded to enhance debugging information in a tablet-like environment.

DebugLog

DebugLog specifies the name of a file to which debug stream data is directed. When you are developing applications, the DebugLog file is a useful place to begin tracing the activity that lead to a crash. By default this is \PENPOINT\LOG on the boot device.

DebugLogFlushCount

DebugLogFlushCount allows you to specify the number of characters that can be written to the debugger stream buffer before it is flushed. Normally this line is commented out. The default value is arbitrarily large, depending on current usage of the file system. However, if you are debugging a program that crashes after a write to the debugger stream, yet before the stream gets written to the log file, you can set this to a small value. When the value is set to 0, text is flushed as soon as it is written to the debugger stream.
DebugSet

DebugSet allows you to set one or more debugger flags to affect the behavior of applications. To specify a flag, follow the string "/D" with the flag identifier. For example, you specify the debug flag B800 with the line:

dbgset=/DB800

To specify more than one flag, separate the flags with one or more spaces. You must specify all flags within one DebugSet line; if your ENVIRON.INI file contains more than one DebugSet line, PenPoint uses the first one in the file.

See \PENPOINT\SDK\INC\DEBUG.H for a list of what subsystem uses what flag. The header files for many subsystems list the effects of setting the various bits in the flags.

Other interesting flags are:

*1 When used with the _PP_OS (debug version of the PenPoint kernel),
directs the heap manager to validate the heap after allocate and free calls
(at a performance cost of about 15 percent). This flag is on by default in
ENVIRON.INI.

D10000 Disables the mini-debugger (production and debug versions of
PenPoint).

D40000 Disables the use of Ctrl+C and Pause keys to exit to the debugger.

G2 Allows page faults during scavenging to enter the debugger.

PenPointPath

Specifies the path to the PENPOINT directory within the boot volume
(theBootVolume). This is useful for maintaining two separate PenPoint systems
on the same hard disk. For example, you might maintain one version of PenPoint
in SYS1\PENPOINT... and another version of PenPoint in SYS2\PENPOINT.... To
use the first system, you would specify PenPointPath=\sys1; to use the second
system, you would specify PenPointPath=\sys2.

PenProxTimeout

PenProxTimeout specifies the number of milliseconds between the pen going out
of proximity and when the out of proximity event is actually sent to the input
system. This delay helps to distinguish between actual out of proximity events and
the user's hand wobbling. The default value is 350 milliseconds.

You can shorten (or lengthen) PenProxTimeout by changing "Gesture Timeout"
from the Settings Notebook. (PenProxTimeout = max(100, GestureTimeout/2).
The default setting of gesture timeout, 600 milliseconds, results in PenProxTimeout
being 300 milliseconds (same as it used to be). Many people can comfortably use the
system with gesture timeout set to .4 seconds, this results in gestures being handled
1/10th of a second faster.
**ScreenHeight**

Specifies the height of the screen in device units.

**ScreenWidth**

Specifies the width of the screen in device units.

**StartApp**

StartApp specifies the first PenPoint application that PenPoint runs when it starts up. Usually this application is the Bookshelf. If you are writing a turnkey application, you might want to name your own application in StartApp, which would make it the only application running on the machine.

**StealMem**

StealMem tells PenPoint to use less memory, so that you can test how your application or service behaves in low-memory conditions. The value specified with StealMem is in hexadecimal. The line `StealMem=100000` causes PenPoint to "lose" 1MB of memory.

**SwapBoot**

SwapBoot allows you to tell PenPoint to boot by reloading its memory from information in the swap file, rather than reloading code. This is called a "swap boot."

To enable swap booting, add `SwapBoot=2` to ENVIRON.INI.

**SwapFileSize**

SwapFileSize specifies the size of the swap file in hexadecimal bytes. The suggested size is 5 MB (or 500000 hex). Be careful when reading this number; 500000 hex equals 5,000,000 decimal. The swap file is created in `\BOOT\PENPOINT.SWP`. If SwapFileSize is set to 0, or is not specified, PenPoint does not create a swap file.

PenPoint swaps pages of data to the swap file. Code is not swapped, it is loaded from the loader data base or boot volume as needed.

**TZ**

TZ specifies the time zone with a string formatted according to the TZ environment variable used in the ANSI time routines. For complete information about the syntax of the TZ line, see Section 1.4, The "TZ Environment Variable," in the *Watcom C Library Reference for PenPoint*.

**Version**

Specifies the PenPoint version label. This label is created by GO and contains copyright and version information. Do not change this line.
VolSel

VolSel specifies the label of the volume that will contain the PenPoint directory hierarchy. You can see the label of a DOS volume when you use the DIR or CHKDSK commands.

From applications you can find the volume specified in VolSel with the well-known UID theSelectedVolume.

If your ENVIRON.INI file does not specify a VolSel environment variable, PenPoint uses theBootVolume.

If you specified a configuration with the Config option, do not modify VolSel.

WinMode

WinMode specifies the initial orientation of the screen device. There are four possible orientations, which correspond to the four possible rotations of a rectangular screen. The possible values are PORTRAIT, LANDSCAPE, PORTRAIT_REVERSE, and LANDSCAPE_REVERSE. In PORTRAIT orientation, the long axis of the screen is vertical; in LANDSCAPE orientation, the long axis of the screen is horizontal. In the _REVERSE orientations, the screen image is rotated 180 degrees from the normal orientation.

You usually use LANDSCAPE when running PenPoint on a PC.

ZoomMargin

ZoomMargin specifies the distance from the bottom of the screen to the bottom of the document opened with AutoZoom.

If you set ZoomMargin to 0 and specify ZoomResize as FALSE, the document that is opened with AutoZoom is locked into the page. If the document is a notebook, the user will not be able to close or resize the notebook. If the document is a page-based application, the user will not be able to turn away from the application.

ZoomResize

ZoomResize specifies whether the document opened with AutoZoom has a resize tab. Possible values are TRUE and FALSE. TRUE means that the document has a resize tab; FALSE means that the document does not have a resize tab.

If you set ZoomMargin to 0 and specify ZoomResize as FALSE, the document that is opened with AutoZoom is locked into the page.

Running in Tablet-Like Mode

When you set Config=Tablet, PenPoint runs in a configuration similar to that of the initial NCR and IBM pen computers. However, there are some considerations to using this configuration:
• **theSelectedVolume** is the drive on your PC from which you booted PenPoint. The bookshelf’s contents live in \PENPOINT\SYS (its DOS name is \PENPOINT\SS) on the hard disk. Installed services, fonts, configured service instances, and so on are also stored on hard the disk.

• PenPoint won’t let you browse theSelectedVolume in the directory view of the Connections notebook. In production PenPoint this prevents users from inadvertently moving, renaming, or deleting files that should be known only to PenPoint. If you want to see the hard disk in the directory view, set debug flag B to 800.

• Similarly, the Installer in the Settings notebook cannot see the hard disk; however, you can still install applications using the Connections notebook.

Before you run in Config=DebugTablet or Tablet mode, you need to:

• Remove the \PENPOINT\SS directory hierarchy from your hard disk. This gets rid of PenPoint state saved previously.

• Delete \PENPOINT\PENPOINT.IDX.

• Delete \PENPOINT\PENPOINT.DIR.

The first time you run in this configuration, you won’t have any of these files.

Now boot PenPoint. The first time you boot, PenPoint will cold boot as usual. However, the loader will build its loader directory under \PENPOINT\SS\LR, which slows booting considerably. PenPoint marks the file system so that the next time you run PenPoint, a warm boot will occur. All subsequent times, booting will be faster.

Note that under this configuration, all installed EXEs and DLLs come from \PENPOINT\SS\LR. If you modify a DLL or EXE and want to replace it, you will either have to start with a clean disk, use the Upgrade utility, or copy your file into the \PENPOINT\SS\LR directory.

If you run in a tablet-like configuration and then decide to perform a cold boot or go back to the RAM configuration, you must clean up your disk. To do this:

• Delete \PENPOINT\PENPOINT.DIR.

• Delete \PENPOINT\PENPOINT.IDX.

• Remove the directory tree under \PENPOINT\SS.

• Remove the directory tree under \PENPOINT\SI.

• Remove the directory tree under \PENPOINT\OI.

On an actual tablet, when PenPoint loads code (services and applications) into the loader database, it deletes the \PENPOINT\APP directory and all application directories underneath it.

Even if you set Config=Tablet, the PenPoint running the SDK does not do this. If it did delete \PENPOINT\APP, it would mean you would have to reinstall the SDK to get \PENPOINT\APP back!
On a real tablet computer, the B2 debug flag controls this. Do not set B2, unless you want to reload the SDK. You can set it in ENVIRON.INI if you really, really want to get a configuration that closely matches a pen computer.

**BOOT.DLC**

The file \PENPOINT\BOOT\BOOT.DLC lists all the system dynamic link libraries that PenPoint needs to operate. A Dynamic Link Library is a body of code that programs can access at run time without needing to be statically linked into each program. It promotes code sharing and reduced executable size. PenPoint uses the same DLL file format and similar technology as OS/2 and Microsoft Windows.

PenPoint takes each name in BOOT.DLC, looks for a .DLL file of the same name in \BOOT\PENPOINT\BOOT\DLL, loads that .DLL and tries to call an initialization routine in the DLL called DllMain. These initialization routines create classes, create objects and perform other initializations. For example, the window system initializes the screen and the input system starts up several subtasks and interrupt subtasks.

There are a standard set of DLLs which implement subsystems of PenPoint. The window system, UI Toolkit, handwriting translation, Application Framework, search and replace, and so on, are all implemented as DLLs.

PenPoint loads and initializes each DLL in order, so the ordering of BOOT.DLC is significant.

The main reason for modifying BOOT.DLC is to enable DB, the PenPoint source-level debugger. The first two lines in BOOT.DLC load the DLLs for DB:

```plaintext
#go0-cdb0-v2 (0) \boot\penpoint\boot\dll\cdb0.dll
#go-cdb3-v2 (0) \boot\penpoint\boot\dll\cdb3.dll
```

If you want to use DB, uncomment both these lines. CDB0.DLL is the Ring 0 portion of the debugger; CDB3.DLL is non-privileged code.

If you are only testing a restricted configuration, you can remove some DLLs; however, this can lead to unpredictable results.

If you have a DLL that you want loaded independent of your application, you can mention it in \PENPOINT\BOOT\BOOT.DLC. This should be rare: only system-wide DLLs such as input, windows, and so on, need to be listed in here.

The order of DLL loading can be significant, so it's best to put your DLL at the end of the list.

You should not need to put your application's DLL file in BOOT.DLC, because your DLL is only needed if the user installs one or more applications which require it. Instead, you put a .OLC file in your application's directory which tells PenPoint which DLLs the application needs.
If you have a DLL that you want loaded independent of your application, you can mention it in `\PENPOINT\BOOT\BOOT.DLC`. This should be rare: only system-wide DLLs such as input, windows, and so on, need to be listed in here. The order of DLL loading can be significant, so it's best to put your DLL at the end of the list.

**CONSOLE.DLC**

The file `\PENPOINT\BOOT\CONSOLE.DLC` allows you to see debugging information while booting PenPoint on a single-screen system. Once PenPoint is booted, you can use the System Log application to view debugging information. You will want to use CONSOLE.DLC when debugging the behavior of your application during a boot—before you can run the System Log application.

To use CONSOLE.DLC, modify your MIL.INI file so that it specifies `MonoDebug=off`.

If the line is commented out, doesn’t exist, or specifies anything other than “off,” debugging information is not sent to your single screen.

**SYSCOPY.INI**

Once PenPoint knows the location of the `BootVolume` and the `SelectedVolume`, it can copy files from other disks to the directory that contains files for the running system. The file SYSCOPY.INI tells PenPoint which files to copy and where to place them.

The files listed in SYSCOPY.INI are those whose locations should not be hard-coded into PenPoint. Because the locations of these files are specified in SYSCOPY.INI, all you have to do is modify SYSCOPY.INI to use a different file. SYSCOPY.INI specifies the locations of the files listed in Table 3-6.

<table>
<thead>
<tr>
<th>File</th>
<th>Used For</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSAIP.INI</td>
<td>List of system applications to load at boot time (non-deinstallable).</td>
</tr>
<tr>
<td>APP.INI</td>
<td>List of general applications to load at boot time (deinstallable).</td>
</tr>
<tr>
<td>SERVICE.INI</td>
<td>List of services to load at boot time.</td>
</tr>
<tr>
<td>PENPOINT.RES</td>
<td>The PenPoint resources file.</td>
</tr>
<tr>
<td>DICT</td>
<td>The dictionary files to load at boot time.</td>
</tr>
<tr>
<td>FONT</td>
<td>The font files to load at boot time.</td>
</tr>
<tr>
<td>PREFS</td>
<td>The preferences to load at boot time.</td>
</tr>
<tr>
<td>HWXPROT</td>
<td>The handwriting prototypes to load at boot time.</td>
</tr>
<tr>
<td>PDICT</td>
<td>The personal dictionary files to load at boot time.</td>
</tr>
</tbody>
</table>
**SYSAPP.INI**

The file `\PENPOINT\BOOT\SYSAPP.INI` lists the system applications that must be present for PenPoint to run correctly. This list includes:

- \penpoint\boot\app\Section
- \penpoint\boot\app\Table of Contents
- \penpoint\boot\app\Notebook
- \penpoint\boot\app\MiniText
- \penpoint\boot\app\Settings
- \penpoint\boot\app\Helpnb
- \penpoint\boot\app\inboxnb
- \penpoint\boot\app\oboxnb
- \penpoint\boot\app\statnb
- \penpoint\boot\app\Connections
- \penpoint\boot\app\Keyboard
- \penpoint\boot\app\Placeholder
- \penpoint\boot\app\Accessry

Applications loaded with SYSAPP.INI cannot be deinstalled. If you are providing a turnkey system, you can load your specific applications from SYSAPP.INI.

**APP.INI**

The file `\PENPOINT\BOOT\APP.INI` specifies the installable applications to load at boot time. These applications can be deinstalled at a later time.

When testing an application, it is usually easiest to add the application to APP.INI. That way the application is installed each time the system is booted.

**Setting Up Specific Configurations**

The preceding sections described the initialization files individually. However, when you change your configuration, you will need to make changes to several files.

This section describes the modifications that you make to support various devices.

**One or Two Monitors**

When you run PenPoint on a PC, you must have a VGA display on your machine. PenPoint can display the entire Notebook user interface on the VGA monitor.

If you have two monitors (one VGA and one monochrome), PenPoint detects that there are two monitors (at boot time) and will simultaneously display the PenPoint Notebook User interface on the VGA screen while displaying debugging output on the monochrome display.

If you have a single monitor and need to view debugging output while booting PenPoint, add a line to your MIL.INI that specifies `MonoDebug=off`. Such a line exists in the _MIL.INI shipped with the PenPoint SDK, but it is commented out.
Configuring a Mouse

The SDK version of PenPoint is set up for a Wacom pen tablet. If you have a mouse, you must modify your MILINI file to specify a mouse. PenPoint supports four types of mice: Microsoft compatible mouse, Logitech C7/C9 mouse, bus mouse, PS/2 mouse.

To attach a Microsoft mouse, add a UNIPENTYPE=Microsoft line to your MILINI, specifying the serial port to which the mouse is connected. To attach a Logitech mouse, add a UNIPENTYPE=Logitech line to your MILINI, again specifying the serial port to which the mouse is connected.

To use a PS/2 mouse, add the line PS2Mouse=on to your MILINI.

Finally, if you are left handed, you might want to add the line LeftyMouse=true to your MILINI.

Configuring a Digitizing Tablet

To use a digitizing tablet with PenPoint, you must:

- Add a line to your MILINI that specifies the tablet and the serial port to which the tablet is attached.
- Configure the serial port with a DOS MODE command before running PenPoint. If the digitizing tablet is attached to COM1, the mode command is:

```
MODE COM1 96,0,7,2
```

Booting PenPoint on a PC

To start PenPoint, run GO.BAT by typing `\penpoint\sdk\util\dos\go`.

GO.BAT is a batch file which sets up your machine and then runs PPBOOT.EXE.

You may need to modify GO.BAT to:

- Configure a serial port for the digitizing tablet.
- Load a Logitech or Microsoft mouse driver.
- Configure a serial port for low-level MIL debug output.
- Unload memory managers and TSRs.
- Run some disk mirroring or FAT preserver utility.

GO.BAT contains a number of lines that are commented out to remind you of these possible additions.

If you are testing a new application and the application goes badly wrong, PenPoint may damage the root directory of your boot volume. This is more likely to be a problem if you run with Config=DebugTablet or Tablet (so that the selected volume set to your hard disk). You may want to use the Norton Utilities' format recover (fr) with the save (/save) option or some other format save utility to save your hard disk's format information before starting PenPoint. If you have this utility, add `fr /save` before `ppboot` in GO.BAT and `chkdsk` after.

If CHKDSK reports an error, run `fr` to fix it.
PC Tools has a similar MIRROR capability.

The hierarchy of applications on the bookshelf files its state in \PENPOINT\SYS\BS on the selected volume. If you are running PenPoint on a PenPoint computer, or if you set VolSel to a disk on a PC, the previous state is still around. The previous configuration of the Notebook should reappear. This is what happens when you power off and power on a PenPoint computer.

If you are running PenPoint on a PC from memory, there’s nothing in memory when you boot PenPoint. Therefore \PENPOINT\SYS will be blank. However, during boot, the default \PENPOINT\BOOT\SYSCOPY.INI instructs PenPoint to copy \PENPOINT\SYS from the boot volume to the selected volume. Thus you can set things up so that a previous saved configuration of the Notebook reappears when you boot from disk. This is how the default Notebook appears when you boot a PenPoint computer—Notebook state was filed and then copied to the boot diskettes.

Loading Debug PENPOINT.OS

GO.BAT runs \PENPOINT\BOOT\PPBOOT, which in turn loads MIL.OS, MIL.INI, and PENPOINT.OS. If you want to load core PenPoint with DEBUG information in it (to get, for example, symbolic names for Class Manager objects), you need to comment out the line

    \penpoint\boot\ppboot \penpoint\boot

and uncomment the line

    \penpoint\boot\ppboot \penpoint\boot\pp.os \penpoint\boot\mil.os \penpoint\boot\mil.ini

What Happens During Booting

PPBOOT prints the names of the files that it loads.

If you’ve enabled LowLevelDebug in MIL.INI, you’ll see debugging information on your second monitor or serial port.

The MIL prints out some memory statistics and then attempts to load PenPoint.

The boot program displays a PenPoint logo and pen. It displays numbers in the lower corners of the screen:

- In the lower left is the version of the PPBOOT boot program.
- In the lower right is a number indicating whether PenPoint found a swap file:
  0  No swap file found.
  1  Swap file found but the boot program couldn’t use it (delete \PENPOINT.SWP and try again).
  2  Old-format swap file found (historical, you should never see this).
  3  Found a swap file, using it.
As PenPoint boots, the pen fills with black. After a while, PPBOOT transfers control to PenPoint, which proceeds to read ENVIRON.INI, BOOT.DLC, SYSAIP.INI, and so on.

After a while (two to four minutes on a 386 PC) you should see the small clock appear, followed by the standard bookshelf and Notebook configuration.

If PenPoint doesn’t work, press \[Pause\] then enter \[q\]. Look in \PENPOINT\LOG on your boot volume for errors.

**Boot Error Messages**

Here are some of the errors that you will probably see during booting, together with explanations:

The first time PenPoint runs, it will complain about a missing PENPOINT.SWP file. This is normal—the swap file does not exist until the second time you run PenPoint.

If you load _PP.OS, you get some additional warning messages about No init routine provided for file.dll.

You might see several Int w/o RB: 7 messages during booting. These are related to parallel port interrupts, are relatively benign, and are explained later in this chapter.

To save memory, PenPoint compacts processes (such as the processCount 0 appMonitor processes). Messages about this are not in error.

**Broken Pen During Booting**

If there’s a problem during booting, a broken pen icon appears on the PenPoint boot screen. The most common value is 1000, which means “you haven’t enabled any pointing device in your MIL.INI.”

The error codes are in the file \PENPOINT\SDK\INC\BOOTERRS.H as shown here:

```
Error codes used with the AbortBoot routine.

These codes are displayed when a fatal error occurs during booting.

Code zero is never used.

Codes 1 through 999 are reserved for the boot program. Of these 1 through 99 are defined by GO and consistent across all machines. Codes 100 to 999 can be used by OEMs for machine specific errors.

The portable codes used by PenPoint start at 1000.

Naming convention: to make it easier to track-down the location of the error code a name is constructed thusly:

"bootErr" + <file name> + <terse description>

See below for examples.
```
Errors in the boot program

#define bootErrDriveNotFound 1
#define bootErrOpenDrive 2
#define bootErrLabelNotFound 3
#define bootErrCorruptFileSystem 4
#define bootErrMILNotFound 5
#define bootErrMILReadErr 6
#define bootErrMILWrongFormat 7
#define bootErrMILWrongSignature 8
#define bootErrMILWrongTailLen 9
#define bootErrMILZeroLength 10
#define bootErrMILTTooBigForMem 11
#define bootErrMILIniNotFound 12
#define bootErrPPNotFound 14
#define bootErrPPMILVersionMismatch 15
#define bootErrPPReadErr 16
#define bootErrPPWrongFormat 17
#define bootErrPPWrongSignature 18
#define bootErrPPWrongTailLen 19
#define bootErrPPZeroLength 20
#define bootErrPPTooBigForMem 21
#define bootErrSwapFileReadErr 22
#define bootErrMemSizing 23
#define bootErrNoLinearPgDirEntry 24
#define bootErrInitFileSys 25
#define bootErrUnknownDrive 26
#define bootErrExitProgram 27

Errors in PenPoint

// Used in DrvMILSetWellKnownIds()
#define bootErrDrvMILNoStylus 1000
#define bootErrDrvMILNoPowerDev 1001
#define bootErrDrvMILNoTimer 1002
#define bootErrDrvMILNoClock 1003
#define bootErrDrvMILNoIntrCtlr 1004
#define bootErrDrvMILNoScreen 1005
#define bootErrDrvMILNoBlock 1006

// File system related
#define bootErrStartupNoBootDiskLabel 1007

// Non-maskable interrupt
#define bootErrDrvMILNMI 1008

// StdError: Note also displays
// TagAdmin(status) and TagNum(status),
#define bootErrStdErrorOccurred 1009

// Environ.ini file not found
#define bootErrStartupNoEnvironIni 1010

// Boot volume not found (or error accessing)
#define bootErrFsvolNoBootVolume 1011

// Invalid PenPointPath specified in environ.ini
#define bootErrFscInvalidPenPointPath 1012
Booting PenPoint on a PC

If code calls StdError, StdSystemError, StdUnknownError or StdMsg before booting is complete, the broken pen displays with the error number of the standard error. The root window isn’t displayed until booting is complete, so the notes that these normally put up are not visible.

Broken Pen Error 1008

Error 1008 indicates that a non-maskable interrupt (NMI) occurred and was not handled by a MIL NMI device. On a PC NMI is used for two purposes: memory parity error and (in some cases) emulation on video cards.

A parity error indicates the failure of a memory chip (soft or hard). You should run a memory diagnostic when the machine is good and hot—partially (30%) blocking the fan is a way to do this. The other possibility is that you are using an older ATI video card. They emulate other types of video cards by generating an NMI when the (nonexistent) registers of the emulated card were accessed. The NMI service routine then emulates the hardware in the video BIOS. The ATI card has a switch which “disabled advanced video modes” and prevented these NMIs. Check to see if your video card uses NMI or supports some form of “emulation.”

When the NMI occurs is a big clue as to what was tickled to cause it (for instance, what DLL was loading, and so on).

Broken Pen Errors between 100 and 999

You may see other boot error numbers, such as 115 or 126, that are not defined in BOOTERRS.H. The boot program overloads the error number by tacking on the number of the drive it is booting from. 0 for floppy and 100 for hard disk. Thus 115 means error 15 when booting from the hard drive.
Using PenPoint on a PC

For more information on working in PenPoint, read the end-user documentation.

Using a Mouse

The Pen can detect not only stylus tip up and down, but also moving the pen in and out of proximity. Handwriting and gesture translation software uses this to figure out when to convert ink dribbles. The mouse does not have any notion of in and out of proximity, so to simulate going out of proximity, you click the middle button. In general, after handwriting or making any gesture with the mouse, “lift the pen away” by clicking middle.

Parallel Port Interrupts

Because of a problem in the 8259 programmable interrupt controller (PIC) used by most PCs, some machines can generate sporadic interrupt 7s under certain, unpredictable conditions. The symptom is that you see several Int w/o RB: 7 messages at boot time. We have not seen this behavior on tablet hardware.

PenPoint attempts to avoid hanging conditions by disabling interrupt 7 whenever too many bad interrupts occur, and the re-enabling interrupt 7 at a later time. This does not limit PenPoint’s functionality.

Theoretically, parallel port I/O could become impossible, if PenPoint were to constantly disable and enable interrupt 7. However, we have not seen this situation.

Installing an Application

While most operating systems require the application developer to provide installation software (or, at a minimum, copy the application and its files to disk), PenPoint provides an Installer, which makes installation of all installable software consistent.

The installer expects application files to be in specific directories on the distribution diskette. The installer copies files from those directories to directories in the RAM file system.

There are two ways you can install an application:

- Use the Installer to install your application while PenPoint is running.
- Add the file name to the APPINI file so that PenPoint installs your application at boot time.

You can also mark a volume so that the Installer is invoked automatically when the disk is mounted. You do this by turning to the disk options card in the connections notebook. When you set Quick Installer to Yes, the Initial View field becomes active. From in this field, you can select whether the installer should be activated for applications, services, and so on when the disk is mounted.
The Installer can do lots of other things for your application, such as copy stationery, help files, bitmaps for icons, and such to the PenPoint computer. The Tic-Tac-Toe application, described in this manual, uses some of these features. These features are also documented fully in Part 12: Installation of the PenPoint Architectural Reference.

Installing an Application While PenPoint is Running

PenPoint is designed to not need rebooting. Users can go forever without having to reload PenPoint and applications from disk. Thus, there needs to be some way for the user to install new applications while PenPoint is running. There is: it involves using the Installer application.

The Installer figures out what applications can be installed by searching for subdirectories of \PENPOINT\APP on all known volumes. So, if you copy your application’s executable and its supporting files to a subdirectory of \PENPOINT\APP on some volume, you will be able to install it while PenPoint is running, just as a user who has purchased your application will install it. In the case of EMPTYAPP, create a directory on your hard disk or on a floppy (with a volume label) called \PENPOINT\APP\EMPTYAPP, and copy EMPTYAPP.EXE to it. (The make file for empty app does this for you.)

Open the Connections notebook, turn to the disks page, and tap on your disk. Choose Applications from the View menu. Empty Application should appear in the list of applications on the disk. Tap on the installed box next to Empty Application.
In this figure, the application installer has found an application directory called Empty Application in `\PENPOINT\APP` on the disk labelled PROJ.

The application name is the name of its directory in `\PENPOINT\APP`. PenPoint supports longer file names than DOS. A rule in the Makefile gave the EMPTYAPP directory and .EXE longer PenPoint names using the STAMP utility.

**Boot-Time Install**

You can get your application installed as part of the boot process by adding it to `\PENPOINT\BOOT\APP.INI`.

APP.INI controls which applications are installed during boot. Each line in APP.INI specifies the PenPoint file name of an application directory. You must specify the volume and the entire path to the application directory. However, lines in APP.INI do not specify the executable file itself. PenPoint copies several different files and

---

To speed up the boot process while testing, you can remove applications you don't wish to use from APP.INI (if there are any).
directories from the application directory to the running application directory (such as help, stationery, and so on).

Thus, for your own project Project Scheduler you would add the following to \PENPOINT\BOOT\APP.INI:

\boot\penpoint\app\Project Scheduler

“System” applications are mentioned in a separate file called \PENPOINT\BOOT\SYSAPP.INI. You should be careful when modifying this file—some of these applications are required for the Notebook environment to work.

Whether an application appears in the Accessories notebook or in the Stationery notebook is determined by a flag in the arguments you send to the Application Manager when you initialize your application class (when the user installs your application). For more information, see the description of clsAppMgr in Part 2: Application Framework of PenPoint Architectural Reference, Volume I.

---

Application .DLL and .DLC Files

If you create multiple applications that use the same classes, you can save memory by placing the shared classes in a common DLL (dynamic link library).

When PenPoint is instructed to load an application (either in response to the Installer or when reading APP.INI), it first finds the directory whose name matches that of the application.

PenPoint then looks in the directory for a file with the same name as the directory, but with a .DLC extension. A .DLC file contains a list of executable and DLL files that the application requires. If your application requires a class that is in a common DLL file, it should list the DLL file in its .DLC file.

If PenPoint finds the .DLC file, reads the names of the DLLs in the file and searches for them in the loader database. If a DLL is not found in the loader database, PenPoint loads it. PenPoint also calls DllMain in the new DLL (if it exists), thus providing each DLL with a standard initialization technique. DllMain can create classes, create objects, and perform other initializations. The last line in the .DLC file must list an executable with the same name as the application directory (with the extension .EXE).

If there is no .DLC file, PenPoint then searches the application directory for an executable with the same name as the application directory (with the extension .EXE).

The name of the .DLC must match the PenPoint name of the directory. If you give the application directory a long PenPoint name, but forget to change the .DLC or .EXE name (with the STAMP utility), installation will fail.
Executing the Application

The easiest way to start an application is to create a document for that application from the Notebook and then turn to or float that document. You can create new documents in several different ways:

- Open Accessories, and tap on the icon of an accessory application.
- Choose an application from the Create menu in the Notebook's table of contents. This creates a new document on the Notebook.
- Make a caret $\wedge$ gesture over the Notebook and choose the application from the stationery menu. This creates a new document at the location where you made the gesture.
- Make a caret $\wedge$ gesture over the bookshelf and choose the application from the stationery menu. This creates a new document in the bookshelf.
- Open the Stationery notebook and use the copy gesture $\downarrow$ to copy a piece of stationery to the main Notebook or elsewhere. Some applications provide several different kinds of stationery, so the user can pick a particular kind of document to start from.

Different applications allow different kinds of document creation. (Appearance in the Stationery notebook is controlled by the appMgr.flags.stationery flag, and appearance in the Accessory notebook is controlled by the appMgr.flags.accessory flag; you set these in your application's init routine). It's convenient during testing to allow both kinds of creation.

Volume Selection

On a PC you have a choice. If you select your hard disk, the state of your simulated PenPoint notebook (documents, table of contents, and so on) will be retained after you quit PenPoint or power off, just as on a PenPoint computer. On the other hand, there is a slight risk that during development your applications won't file correctly and hence will cause trouble when you restart PenPoint. Running PenPoint from RAM avoids these problems, but presents you with an empty Notebook every time you restart it.

Interrupting PenPoint

On a PC, the [Pause] key interrupts the current task. It puts you either in a mini-debugger built into PenPoint, or into DB if you are running the PenPoint Source Debugger. In either case, type g to continue or q to quit and return to DOS. (Of course, you can only return to DOS on a PC).
Exiting PenPoint

There are two ways to exit from PenPoint:

- Tap on Settings, navigate to Preferences-Power, and tap the Shutdown button. Confirm that you want to power-off PenPoint.
- Press \Pause\ and enter q.

Note that either way, PenPoint takes time to shut down because it has to flush all cached information to disk before halting.

Depending on your machine configuration, your PC may return to DOS or may hang. GO has found that network software, spoolers, TSR's, disk caches, and RAM drives can all affect this. If your machine does not return to the DOS prompt, reboot.

When you get a DOS prompt, you should run CHKDSK to check for file system errors.

More on the Bookshelf

If you're running PenPoint on a PC and your root volume is your hard disk, there will be lots of files in \PENPOINT\SYS\BS. This directory contains all notebooks and all their files. You should be able to find your documents' directories in BS. Within each document directory, you'll find a DOC.RES file, which contains the document's objects and data.

After exiting PenPoint, you'll find that PenPoint has left PENPOINT.DIR files in \PENPOINT and in its subdirectories. The PENPOINT.DIR files map PenPoint file system names to the DOS files and contains attribute information about the files.

Using the Notebook

The Notebook initially displays its table of contents view. This is always at the front of the Notebook.

You can page through applications by tapping the arrows to the left or right of the page number. You can also flick backwards and forwards through the pages by drawing a short line to the left or right in the title bar.

You can also jump to a page by tapping the button in its page number in the table of contents. If you double-tap on the document's button, it is "torn out" from the Notebook and floats on top of the Notebook (if you enable this in the Float & Zoom portion of Settings).

For more information on using PenPoint, read the end-user manual, Getting Started with PenPoint.
The Universal Serial Pen Driver

After writing a number of PenPoint digitizer drivers for devices that connect through a serial port, it became apparent that they were all pretty much alike. These pointing devices want to pass three or four pieces of information into the computer: x position, y position, tip-down and (optionally) pen-in-proximity-to-screen.

Because this amount of information is difficult to pack into a single byte most serial pointing devices opt for a multi-byte format. Some bit or pattern is reserved to signal the start or end of the multi-byte frame, the tip and proximity states each get a bit somewhere in the frame, and the bits which make up the X and Y position are given as many bits (in as many bytes) as they need. Some devices (like mice) are relative and pass small (less than 8-bit) deltas. In contrast, digitizers return absolute coordinates, which require 10-13 bits of space in binary formats or 4-5 bytes or characters in ASCII formats.

Unfortunately, each manufacturer that needed such a protocol invented their own. Later products (such as the Wacom) can be programmed to emulate many of the pre-existing products and protocols.

The universal pen driver is an attempt to eliminate the need for a custom driver for every digitizer and mouse in the world. This driver parses a simple language that describes the protocol of a serial pointing device. The driver also contains a number of "canned" or predefined descriptions of some popular products.

The key expressions in this language usually contain:

- The relative position of a byte in the frame.
- A mask to AND the byte with.
- A value to compare the result with.

Thus if bit 3 in the second byte is zero when the tip is touching we describe this as T, 1, 4, 0; 1 for the byte (bytes are numbered from 0), 4 for masking bit 3, and 0 as the expected result when the tip is down. The expression P, 0, 9, 8 says that the pen is in proximity to the screen when byte 0 contains: 0 in bit 0 and 1 in bit 3 (1001B mask compared with 1000B).

Numeric values are described in a similar way:

- A byte number
- A mask
- An offset
- A scaling factor

To support values whose bits are split across multiple bytes, UniPen sums the frame expressions that have the same tag. For example, in the Bit Pad 2 binary format, the 12 bits of the X position are contained in the low-order 6 bits (0 through 5) of the second and third bytes (bits 0 through 5 come from bits 0
through 5 of the second byte; bits 6 through 11 come from bits 0 through 5 of the third byte). We describe this with the string:

\[ x, 1, 63, 0, 1 \quad x, 2, 63, 0, 64 \]

The value 63 (0111111B) masks the low-order 6 bits; the scaling factor of 1 is a no-op, the scaling factor of 64 shifts the value 7 bits to the left before summing.

To describe the Bit Pad 2 ASCII format we use the offset to subtract the value of an ASCII "0" to convert a 4-digit ASCII string starting in the first byte:

\[ x, 0, 127, -48, 1000 \quad x, 1, 127, -48, 100 \quad x, 2, 127, -48, 10 \quad x, 3, 127, -48, 1 \]

Other digitizers are left as an exercise to the student.

**UniPen Command Syntax**

Table 3-7 lists the UniPen commands.

<table>
<thead>
<tr>
<th>Command</th>
<th>Meaning</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>V</td>
<td>VERBOSE</td>
<td>Toggles the verbose mode used for debugging. The compile time switch VERBOSE must be enabled in the source code for this to work.</td>
</tr>
</tbody>
</table>
| C,baud,parity,| COMPORT        | Sets the com port settings for the operating mode: baud = the baud rate parity ('O'= Odd, 'E'= even, 'N' = none); bits = the 
| bits,stop     |                | # of data bits (6,7,8); stop = the number of stop bits (1,2).                                                                      |
| I,byte1,byte2,| INIT           | Specifies a sequence of bytes to be sent to the device for initialization. The sequence can be up to 16 bytes long.                     |
| ...            |                |                                                                                                                                         |
| G,baud,par,bit,| ALTERNATEINIT  | Specifies an alternate sequence of bytes to be sent to the device with a different com port setting. This is used when the 
| stop,b1,...    |                | device's initial com port settings are different than the operating settings (cmd-C). baud = the baud rate parity ('O'= Odd, 'E'= even, 'N' = none); bit = the 
|                |                | # of data bits (6,7,8); stop = the number of stop bits (1,2); b1,... = the byte sequence to send. If the baud rate is zero, the sequence will be sent to 
|                |                | the device at 1200,2400,4800, 9600 baud (four times). The alternate init ('G') happens before the INIT command ('I').                  |
| N,name         | NAME           | Specifies a name for the protocol. The name can be any sequence of characters except space or comma. If used, this must be the first command if used. |
| L,xLimit,yLimit| LIMIT          | Sets the maximum value for x and y. These limits must be set properly for the proper operation of the pen. In a machine where the 
|                |                | digitizer is not combined with the screen, these limits are used to scale the pen's position to fit the screen. This allows the use of digitizers and screens of different 
|                |                | dimensions. For combined screen and digitizers (such as the SuperScript), these limits are used for coordinate transformation when different screen orientations are chosen. If you lose 
|                |                | pen and ink alignment when you change the orientation preference, these values are probably incorrect.                                 |

continued
Table 3-7 (continued)

<table>
<thead>
<tr>
<th>Command</th>
<th>Meaning</th>
<th>Definition</th>
</tr>
</thead>
</table>
| T,byte,mask,compare | TIP          | Defines an equation for the computing the tip state where: \( TIP = ((\text{protocol}[\text{byte}] \& \text{mask}) == \text{compare}) \);
byte = the byte index into the protocol stream from 0 to the max byte - 1; mask = the value the byte is masked with; compare = value the masked byte is compared with. |
| P,byte,mask,compare | PROXIMITY     | Defines equation for the computing the proximity state where: \( TIP = ((\text{protocol}[\text{byte}] \& \text{mask}) == \text{compare}); \) byte = the byte index into the protocol stream from 0 to the max byte - 1; mask = the value the byte is masked with; compare = value the masked byte is compared with. |
| X,byte,mask,add,mult | X            | Defines an equation for computing X where: \( X = X + ((\text{packet}[\text{byte}] \& \text{mask}) + \text{add}) \times \text{mult}; \) byte = the byte index into the protocol stream from 0 to the max byte - 1; mask = the value the byte is masked with; add = the value added to the masked value; mult = the value multiplier. There can be more than one X command. |
| Y,byte,mask,add,mult | Y            | Defines an equation for computing Y where: \( Y = Y + ((\text{packet}[\text{byte}] \& \text{mask}) + \text{add}) \times \text{mult}; \) byte = the byte index into the protocol stream from 0 to the max byte - 1; mask = the value the byte is masked with; add = the value added to the masked value; mult = the value multiplier. There can be more than one Y command. |
| S,byte,mask,compare | SYNC         | Defines an equation for checking the validity of the packet. If \( (\text{packet}[\text{byte}] \& \text{mask}) \neq \text{compare} \) the packet is considered out of sync. byte = the byte index into the protocol stream from 0 to the max byte - 1; mask = the value the byte is masked with; compare = value the masked byte is compared with. |
| B,byteCount | BYTECOUNT    | Defines the number of bytes in a packet.                                                                                                                                                           |
| E           | EXTEND       | Tells UniPen to sign extend the X and Y values from 8 bits to 16 bits.                                                                                                                              |
| J,byte,mask,compare | NEGATE X     | Used to process sign bits in the protocol for X. X will be negated if \( (\text{packet}[\text{byte}] \& \text{mask}) == \text{compare} \). byte = the byte index into the protocol stream from 0 to the max byte - 1; mask = the value the byte is masked with; compare = value the masked byte is compared with. |
| K,byte,mask,compare | NEGATE Y     | Used to process sign bits in the protocol for Y. Y will be negated if \( (\text{packet}[\text{byte}] \& \text{mask}) == \text{compare} \). byte = the byte index in the protocol stream from 0 to the max byte - 1; mask = the value the byte is masked with; compare = value the masked byte is compared with. |
| F,[XY]      | FLIP X or Y  | Inverts the coordinates of either X or Y. For example, \( F, X \) would result in \( X = x\text{Limit} - X \) being executed. \( F, Y \) would do the same for Y.                                                             |
### Table 3-7 (continued)

<table>
<thead>
<tr>
<th>Command</th>
<th>Meaning</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>R,num,res,byte1,byte2,...</td>
<td>RESOLUTION</td>
<td>Tells UniPen the resolution of the device and defines a sequence of bytes to initialize the mode. There must be at least one resolution command and there can be as many as three. If there is more than one resolution command, the byte sequence will be sent to the device whenever that resolution mode is to be used. If there is only one resolution command, the byte sequence is optional. num = the resolution number (0,1,2); res = the resolution in counts/meter; byte1,... = the byte sequence used to initialize the mode.</td>
</tr>
<tr>
<td>Q,num,res,byte1,byte2,...</td>
<td>SAMPLE RATE</td>
<td>Tells UniPen the sample rate of the device and defines a sequence of bytes to initialize the mode. There must be at least one sample rate command and there can be as many as two. If there is more than one sample rate command, the byte sequence will be sent to the device whenever that sample rate mode is to be used. If there is only one sample rate command, the byte sequence is optional. num = the sample rate number (0,1,2); res = the number of samples per second; byte1,... = the byte sequence used to initialize the mode.</td>
</tr>
<tr>
<td>D,scale</td>
<td>DELTA</td>
<td>Tells UniPen that the counts from the device are relative positions and not absolute positions. scale = the scale factor for the delta counts. (1 = no scale, 2 = 2x scale, and so on.)</td>
</tr>
<tr>
<td>A</td>
<td>ABSOLUTE</td>
<td>Tells UniPen that the counts from the device are the absolute position of the pen.</td>
</tr>
<tr>
<td>O,x,y</td>
<td>OFFSET</td>
<td>Offsets the x and y positions or defines the difference between logical zero and physical zero. A positive number will bring the &quot;ink&quot; closer to the lower left corner; a negative one farther away.</td>
</tr>
<tr>
<td>H,threshold</td>
<td>THRESHOLD</td>
<td>Defines the reporting threshold. The pen must move at least threshold counts before a new position will be reported. For most devices this can be one.</td>
</tr>
<tr>
<td>U</td>
<td>UNIFIED</td>
<td>The display and digitizer are a combined unit. When they are not, PenPoint scales the digitizer position to fit the screen based on the values provided by the Limit command (see above). Note that for accurate matching of the pen and &quot;ink&quot; you must specify ScreenPixelsPerMeter in MIL.INI to match your screen. This may be easily calculated by measuring the width of the actual pixels along the long (landscape) axis in centimeters. ScreenPixelsPerMeter = 640 / length * 100</td>
</tr>
<tr>
<td>M</td>
<td>MUNGE PROXIMITY</td>
<td>Supresses proximity sensing for any pre-defined type. Note that there is no reason to use this command when defining your own protocol, just don’t specify a “P” tag.</td>
</tr>
</tbody>
</table>
Notes on Using the UniPen Driver

Use the UNIPENPORT tag in MIL.INI to set the communications port. For example, the first line sets UniPen to COM1:, the second line sets UniPen to the communication port at 2F8, IRQ 5:

```
UNIPENPORT = 1
UNIPENPORT = 760,5
```

Use the UNIPENPORT tag in MIL.INI to select a predefined protocol. This line sets UniPen to the Wacom 510C digitizer:

```
UNIPENTYPE = WACOM510C
```

The current predefined types are:

- **MICROSOFT**  Microsoft serial, two-button mouse.
- **LOGITECH**  Logitech C7 or C9 serial, three button mice. For a Logitech MouseMan use the MICROSOFT tag.
- **WACOM510C**  The switches under the control unit must be set as shown here (X = on, O = off):

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>DS1</td>
<td>DS2</td>
<td>DS3</td>
</tr>
<tr>
<td>0X0X0X0X0</td>
<td>0X0X0X0X0</td>
<td>0X0X0X0X0</td>
</tr>
<tr>
<td>0X0X0X0X0</td>
<td>0X0X0X0X0</td>
<td>0X0X0X0X0</td>
</tr>
</tbody>
</table>

- **WACOM510**  The older Wacom units, red power LED, attached power cord. The switches must be set as shown here (X = on, O = off):

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Front</td>
<td></td>
<td>Back</td>
</tr>
<tr>
<td>000X0X0X0</td>
<td>0X0X0X0X0</td>
<td>0X0X0X0X0</td>
</tr>
<tr>
<td>0X0X0X0X0</td>
<td>0X0X0X0X0</td>
<td>0X0X0X0X0</td>
</tr>
</tbody>
</table>

- **SuperScriptII**  The SuperScript II LCD/digitizer combo. For the ink to be aligned you must specify `ScreenType=SuperScript` or `ScreenPixelsPerMeter=3690` in MIL.INI.

- **CalCompDBII**  CalComp's DrawingBoard II, should also work with their "Wiz" product. Use the default "Hi Resolution Binary" format (#23) in "run" mode, 9600 baud no parity, eight data bits, one stop bit, 1000 lpi resolution, 125 pps. Enable "Send when out of proximity," button 10=1 in bank B.

- **AceCat5by5**  A very low cost (under $150) 5" by 5" digitizer from AceCAD. Supports proximity, has corded pen. Draws its power from the PC keyboard connector or an optional external AC adapter.
These are generic descriptions. You will probable have to "tune" these by placing the parameters into MIL.INI and adjusting for the specific characteristics of your digitizer.

**GAZELLE** For products from Gazelle System (now owned by Logitech).

**MM**  The common "MM" digitizer protocol.

**BITPAD2**  BitPad 2 binary protocol.

**BITPAD2ASC**  BitPad 2 ASCII protocol.

To define a new protocol in MIL.INI use the UNIPENPROTOCOL tag. Also, since protocol definitions can get long, we created several tags for defining protocols. It does not matter which commands go with which tags, because all of the tags are sent to one central parser. The tags are UNIPENCOMPORT, UNIPENINITIALIZE, UNIPENRESOLUTION, UNIPENSAMPLERATE, UNIPENTIP, UNIPENPROXIMITY, UNIPENXPROTOCOL, UNIPENYPROTOCOL, UNIPENPROTOCOL. For example, the Microsoft serial mouse protocol is:

```
UNIPENTIP          = T,0,32,32
UNIPENPROXIMITY    = P,0,16,0
UNIPENXPROTOCOL    = X,0,3,0,64 X,1,63,0,1
UNIPENPROTOCOL     = Y,0,12,0,16 Y,2,63,0,1 F,Y
UNIPEN_PROTOCOL    = B,3 S,0,64,64, L,3600,2400 E D,5 H,1
UNIPENRESOLUTION   = R,0,7800
UNIPENSAMPLERATE   = Q,0,30
```

The C,L,T,B,R,Q,H, at least one X and Y, and an A or a D command are required.

**Other Sample Definitions**

`SuperScriptII`

```
I,82,48,61,48,13,10 C,9600,N,8,1 T,0,1,1 P,0,64,0
X,1,3,0,16384 X,2,127,0,128 X,3,127,0,1
Y,4,3,0,16384 Y,5,127,0,128 Y,6,127,0,1
S,0,128,128 B,7 L,6911,5183 A R,0,39370 Q,0,100 H,1 U 0,0,157;
```

`AceCat5by5`

```
C,9600,0,8,1 R,0,19500 Q,0,100 L,2437,2437
I,0,64,98 P,0,64,0 T,0,1,1 X,1,127,0,1 X,2,127,0,128
Y,3,127,0,1 Y,4,127,0,128 S,0,128,128 A H,1 B,5
```

`Gazelle:`

```
N,GAZELLE C,9600,0,8,1 R,0,16185 Q,0,50 T,0,1,1 P,0,64,64
X,1,127,0,1 X,2,31,0,128 Y,3,127,0,1 Y,4,31,0,128
S,0,128,128 L,3200,2530 O,210,300 B,5 A H,10;
```

`MM:`

```
N,MM C,9600,0,8,1 R,0,19500 Q,0,100 L,5000,5000
I,0,64,98 P,0,64,0 T,0,1,1 X,1,127,0,1 X,2,127,0,128
Y,3,127,0,1 Y,4,127,0,128 S,0,128,128 A H,1 B,5;
```
BitPad2ASCII:

N,BITPAD2ASCII C,9600,E,8,1 R,0,7800 Q,0,100
T,10,01,01 P,10,01,01 L,5000,5000
X,0,127,127,100 X,1,127,127,100 X,2,127,127,100 X,3,127,127,100
Y,5,127,127,100 Y,6,127,127,100 Y,7,127,127,100 Y,8,127,127,100
S,11,127,13 B,13 A H,1;

BitPad2:

C,9600,E,8,1 R,0,7800 Q,0,100 T,0,4,4 P,0,1,0
L,5000,5000 X,1,63,0,1 X,2,63,0,64 Y,3,63,0,1 Y,4,63,0,64
S,0,64,64 B,5 H,1 A;

LOGITECH:

C,9600,0,8,1 R,0,7800 Q,0,50 L,3600,2400
P,0,1,0 T,0,4,4 X,1,127,0,1 J,0,16,16 F,X Y,2,127,0,1 K,0,8,8 F,Y
S,0,128,128 D,5 H,1 B,3;

WACOM510C:

C,9600,0,7,2 R,0,9906 Q,0,100 T,0,4,4 P,0,1,0
L,2320,1510 X,1,63,0,1 X,2,63,0,64 Y,3,63,0,1 Y,4,63,0,64
S,0,64,64 B,5 H,1 A;

WACOM510:

C,9600,0,8,1 R,0,9906 Q,0,100 T,0,4,4 P,0,1,0
L,2320,1510 X,1,63,0,1 X,2,63,0,64 Y,3,63,0,1 Y,4,63,0,64
S,0,64,64 B,5 H,1 A;

CalCompDBII:

C,9600,N,8,1 R,0,39370 Q,0,125 T,0,4,4 P,3,32,0
L,7000,5250
X,0,3,0,16384 X,1,127,0,128 X,2,127,0,1
Y,3,3,0,16384 Y,4,127,0,128 Y,5,127,0,1
S,0,128,128 B,6 H,1 A;
Part 2 / Debugging PenPoint Applications
PART 2 / DEBUGGING PENPOINT APPLICATIONS

Chapter 4 / Introduction
DB Overview 4.1 67
Organization of This Part 4.2 67
Sample Files 4.3 68

Chapter 5 / Preparing to Run the Debugger
Files Used in a DB Session 5.1 69
Compiling and Linking 5.2 69
Installing DB 5.3 70
Installing Applications to Debug 5.4 70
Start PenPoint 5.5 70

Chapter 6 / Using DB
Invoking DB: the PAUSE Key 6.1 71
Continuing Execution: the G Command 6.2 71
Module Names, Process Names, and Task IDs 6.3 71
Hexadecimal Numbers 6.4 72
Source Code Debugging 6.5 72
Finding and Loading Symbols: the SYM Command 6.5.1 72
Using Source Code: the SRCDIR Command 6.5.2 72
Setting DB's Context: the CTX Command 6.6 73
Breakpoints: the BP, BL, and BC Commands 6.7 73
Viewing the Call Stack: the ST Command 6.8 74
Frame Numbers and the CTX Command 6.8.1 75
Examining and Setting Values 6.9 75
The ? (Evaluate) Command 6.9.1 75
Known Identifiers: the IDS command 6.9.2 76
Identifier Types: the TYPE Command 6.9.3 76
Lexical Scope 6.9.4 77
Single-Stepping: the p, P, t, and T Commands 6.10 77
Viewing Source Code: the V Command 6.11 77
Viewing Assembly Code: the U Command 6.12 78
Executing C Code 6.13 79
Executing Code at the DB Prompt 6.13.1 79
Executing Code at a Breakpoint 6.13.2 79
Task List: the TL Command 6.14 80
Saving Typing 6.15 80
DB.INI File 6.15.1 80
Using DB Scripts 6.15.2 81
Command Line Editing 6.15.3 81

Chapter 7 / DB Command Reference
Command Summary 7.1 85
Notation Conventions 7.2 86
Scope Specification (scopeSpec) 7.2.1 86
Line Count (lineCount) 7.2.2 88
Code Addresses 7.2.3 88
Data Addresses 7.2.4 88
Line Numbers 7.2.5 88
“Scope.Identifier” Referencing 7.2.6 88
Task Set (taskSet) 7.2.7 89
Command Datasheets 7.3 89

Chapter 8 / Profiling with DB
Profile Breakpoints 8.1 113
Two Types of Profiles 8.2 113
Code Profiling 8.3 113
Code Profiling Options 8.3.1 114
A Caveat Concerning Sampling Profiles 8.3.2 115
Getting More Frequent Samples 8.3.3 115
Code Profiling Examples 8.3.4 115
Object Profiling 8.4 117
Basic Object Profiling 8.4.1 117
Object Profiling Options 8.4.2 117
Object Profiling Message Pattern 8.4.3 117
Object Profiling Examples 8.4.4 118
Displaying Profile Information: the DP Command 8.5 119
DP Command Examples 8.5.1 119
Clearing Profile Information: the ZP Command 8.6 121
Profiling Specific Tasks 8.7 121
Chapter 9 / Advanced DB Techniques

Skipping Execution 9.1 123
Controlling Threads of Execution 9.2 124
The FZ ("freeze") Command 9.2.1 124
The TH ("thaw") Command 9.2.2 124
The TT ("terminate task") Command 9.2.3 124
Commands Executed at Compile Time 9.3 124
DB Built-Ins 9.4 125
DB's Predefined Types 9.4.1 125
Useful Values in DB 9.4.2 126
DB's Useful Variables 9.4.3 127
DB Runtime Routines 9.4.4 128
The ON Command 9.5 129
Program Events 9.5.1 129
Access Events 9.5.2 129
Task Events 9.5.3 129
Fault Events 9.5.4 130
Other Events 9.5.5 130
The INSTALL and START Commands 9.6 131
Context Inside of Breakpoints 9.7 131
Cast Operator 9.8 131
Tilde Operator 9.9 131

Chapter 10 / General PenPoint Debugging Techniques

DEBUG Compiler Option 10.1 133
PenPoint Uses DEBUG 10.1.1 133
Using DEBUG in Your Programs 10.1.2 134
Debug Versions of PenPoint DLLs 10.1.3 134
Debugging Flag Sets 10.2 134
The Debugger Stream 10.3 135
Configuring the Debugger Stream Destinations 10.3.1 135
Writing to the Debugger Stream 10.3.2 137

Chapter 11 / The System Log Application

Loading the System Log Application 11.1 141
Running the System Log Application 11.2 141
System Log Application Menus
  Show Menu 11.3.1 142
  Trace Menu 11.3.2 142
  Log Size Menu 11.3.3 143
  Font Menu 11.3.4 143

Chapter 12 / PenPoint Mini-Debugger

Invoking the Mini-Debugger 12.1 145
Mini-Debugger and DB 12.1.1 145
On a PenPoint Computer 12.1.2 146
Mini-Debugger Commands Setting Debug Flags 12.2.1 147
Using the Mini-Debugger 12.3 148
Map Files 12.4 148
Exception Handling 12.5 148
Understanding Interrupts 12.6 148
The Task List 12.7 151

List of Figures
11-1 System Log Application on a PC 142

List of Tables
5-1 Files Used in a DB Session 69
7-1 DB Command Summary 85
7-2 Scope Specification 87
7-3 Specifying a Task Set 89
8-1 Routine Set Specification 114
8-2 Message Pattern Specification 117
8-3 msgList Specification 118
8-4 objectList Specification 118
8-5 DP Flags 119
9-1 DB's Predefined Types 125
9-2 Useful Values in DB 126
9-3 DB's Useful Variables 127
10-1 Explicit Writes to the Debugger Stream 137
10-2 Warning Message Passing Macros 138
10-3 Message Passing Functions 138
10-4 Expression Handling Macros 139
12-1 Mini-Debugger Commands 146
Chapter 4 / Introduction

The PenPoint™ operating system provides several tools that you can use to debug PenPoint programs. These include:

- A source level debugger.
- A debugging stream that can be written to a log file, viewed on a second monitor (when running PenPoint on a PC), and viewed in the PenPoint System Log application.
- Functions that write text to the debugging stream, such as DPrintf() and Debugf().
- Macros that can write text to the debugging stream when messages return warning or error status values.
- A low-level mini-debugger.

Most of this part is dedicated to the PenPoint source level debugger, DB. The other debugging facilities are described in Chapters 10, 11, and 12 of this part.

DB Overview

DB, helps you to trace program execution and measure performance. DB has several features that allow you to interact with PenPoint at the source code and assembly language levels:

- Full support of PenPoint's multi-tasking environment. You can switch easily among different tasks to examine their states and control their execution.
- Command line evaluation of declarations and expressions. You can enter C declarations and expressions directly from the command line, allowing you to alter code for "what-if" trials.
- Batch command file and execute-on-break stored command sequences. You can build powerful debugging scripts and macros to speed up complex debugging operations.

Organization of This Part

Chapter 4, this chapter, introduces the various debugging tools and describes the rest of the part.

Chapter 5, Preparing to Run DB, describes how to compile and link your programs so that you can view symbolic names and locations in your source code while running DB.

Chapter 6, Using DB, provides a tutorial on how to use DB to perform most common debugging procedures.
Chapter 7, DB Command Reference, provides a reference to all of the DB commands.

Chapter 8, Profiling With DB, describes how to use DB's powerful profiling capabilities.

Chapter 9, Advanced DB Techniques, provides a tutorial on using some of DB's advanced features, which include: controlling threads of execution; using DB's built-in types, variables, and routines; and setting breakpoints on events.

Chapter 10, General PenPoint Debugging Techniques, describes other debugging tools and techniques.

Chapter 11, The System Log Application, describes a PenPoint application that you can use to view the debugger stream, set DEBUG flags, and view other system data.

Chapter 12, PenPoint Mini-debugger, describes the mini-debugger and the differences between it and DB.

**Sample Files**

This manual uses the CALC sample application in most of its examples. The source code for CALC and several other sample applications is in `\PENPOINT\SDK\SAMPLE`. 
Chapter 5 / Preparing to Run the Debugger

Files Used in a DB Session

In addition to the program you are debugging, DB uses several other files to configure itself. Table 5-1 lists files that are involved in a debugging run. This list includes both the files that you edit and compile, and files that the PenPoint™ operating system and DB access during execution.

Table 5-1
Files Used in a DB Session

<table>
<thead>
<tr>
<th>File</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>DB.INI</td>
<td>Optional file. If it exists, DB will interpret the contents of the file when it starts up.</td>
</tr>
<tr>
<td>CDB3.DLL</td>
<td>The user-mode portion of the PenPoint load module for DB.</td>
</tr>
<tr>
<td>CDB0.DLL</td>
<td>The supervisor-mode portion of the PenPoint load module for DB.</td>
</tr>
<tr>
<td>ENVIRON.INI</td>
<td>PenPoint reads this file at cold boot to configure the general execution environment. You can also set debugging flags in it, and specify the location of DB.INI.</td>
</tr>
<tr>
<td>*.C, *.H</td>
<td>The source files for your program. The Appendices contain source code for sample applications.</td>
</tr>
<tr>
<td>YOURPROG.EXE</td>
<td>When your compiled program has successfully linked with the PenPoint libraries the result is this PenPoint loadable module.</td>
</tr>
<tr>
<td>YOURPROG.DLL</td>
<td>Dynamic Link Libraries (.DLLs) are used to store shareable portions of your executable code.</td>
</tr>
</tbody>
</table>

Compiling and Linking

To run under DB, you follow the same compiling and linking steps covered in Chapter 6 of the Application Writing Guide. There are a few additional steps that you have to take to cause the compiler and linker to generate symbolic debugging information.

The WATCOM C/386 compiler and linker can generate two levels of symbolic debugging information.

- The first level of information includes line numbers and symbols for public routines. To get this level of information:
  - When compiling, set the compiler’s /Of+ and /D1 switches.
  - When linking, include the line DEBUG LINES or DEBUG ALL before the FILE line(s) in your link command file.
The second level of symbolic debugging information includes full symbol name and type information. To get this level of information:

- When compiling, set the compiler's `/Of+` and `/D2` switches.
- When compiling, be sure not to set any of the `/Oa` switches.
- When linking, include the line `DEBUG ALL` before the `FILE` line(s) in your link command file.

In all cases, you should set the compiler's `/en` switch if you want DB's `st` command to display routine names for modules with no symbols loaded.

You can create different levels of symbolic debugging information for different source files.

### Installing DB

DB is a PenPoint DLL. To install DB, remove the `#` from the following line in `\PENPOINT\BOOT\BOOT.DLC`:

```plaintext
#go-cdb0-v2(0) \boot\386\penpoint\boot\dll\cdb0.dll
#go-cdb3-v2(0) \boot\386\penpoint\boot\dll\cdb3.dll
```

### Installing Applications to Debug

You install the program that you wish to debug in the normal fashion. The changes to it to support debugging are in compiling and linking, not in installing.

### Start PenPoint

Start PenPoint in the normal fashion.
Chapter 6 / Using DB

This chapter introduces you to DB’s most important concepts and operations.

Invoking DB: the PAUSE Key

To switch from the PenPoint™ operating system UI interaction to DB interaction, press the Pause key on the keyboard. The current application process is frozen and the DB prompt appears.

The prompt is a greater-than character (> ) followed by a tiny dot. When you type a command into DB, the command is displayed immediately to the right of the prompt:

> tl

The command is sent to DB when you press the Enter key.

Continuing Execution: the G Command

To continue PenPoint execution, type the g command.

Module Names, Process Names, and Task IDs

PenPoint executable modules (.DLLs and .EXEs) are given names when they are linked. The convention for these module names is:

    companyName-moduleName-majorVersion(minorVersion)

DB gives PenPoint application processes names which are a variation on these module names. The name of a process is simply the quoted name of the module that contains the entry point for the process and followed by a number in square brackets. For instance:

"go-calculator-v1(0)"[1]

The number in square brackets indicates the process count of the application. Process count 0 is the application manager process, which owns the installed application class itself. The kernel increments an application’s process count each time that application is run; process count values greater than zero indicate how many times the application has been executed (since the last cold boot).

PenPoint tasks have 16 bit task IDs. (See Chapter 75 in Architectural Reference Manual Part 8: System Services.)
Hexadecimal Numbers

A hexadecimal number starts with a zero (0) and uses the hexadecimal numerals 0–9 and a–f.

DB prints the leading zero in its output (except in obviously hexadecimal cases, such as dumps). DB requires that you preface all input hexadecimal numbers with a 0. 01db8 is a typical hexadecimal number.

Source Code Debugging

To debug source code, DB needs two pieces of information:

- DB needs to read symbolic debugging information. Compilers and linkers put symbolic debugging information in the .EXE or .DLL file on disk. So you may need to tell DB:
  - Where to find the .EXE or .DLL file.
  - Which .EXE or .DLL symbolic information to read.
- DB needs to know where to find the source code files.

Without this information DB can still be used, but only at the assembly code level. The next two sections tell you how to provide this information to DB.

Finding and Loading Symbols: the SYM Command

Use the sym command to tell DB to read symbolic debugging information into memory:

```bash
>sym "calculator"
>sym "calc_eng"
```

If the module doesn’t contain symbolic debugging information (probably because it was compiled or linked without the appropriate flags), DB displays a warning. See Section 6.16.3 for information on sym command options that can reduce the amount of memory required.

Using Source Code: the SRCDIR Command

To display source code and to allow DB to display source code (e.g. after taking a breakpoint) DB must be told where source files are located.

DB’s srmdir command tells DB where to find the source files that correspond to a given module. The volume name (after the \) must be the DOS volume name. You can see the DOS volume name with the DOS DIR or CHKDSK command; you can modify it with the DOS LABEL command.

```bash
>srmdir "calculator" \c\penpoint\sdk\sample\calc
>srmdir "calc_eng" \c\penpoint\sdk\sample\calc
```

Alternatively, you can set the DBSrc environment variable in \PENPOINT\BOOT\ENVIRON.INI. This is particularly useful if you commonly work with source files.
in many directories. DB will search these directories after searching any directories specified in srcdir commands.

DBSrc=\c\dir1;\c\dir2;\c\dir3

## Setting DB's Context: the CTX Command

DB is always focused on some area of the program that you are debugging. This focus is called the context. The context includes:

- A task ID (the current task) and its current code address.
- A stack frame of a currently active function call (the current call).
- A name scope in which to look up identifiers (the current scope).

The ctx command sets DB's context. DB suspends the task and sets the current call and current scope, based on the task's current code address.

To set the context to a particular task, type the following. (DB's tl command lists all tasks.) When you set DB's context to a given task, DB's prompt changes to include the process name and ID of that task. In this example, "go-calculator-v1(0)"[1] is DB's name for the process, and 0508 is PenPoint's task ID.

```
>0508 ctx
"go-calculator-v1(0)"[1] 0508>
```

To set the context to the executable module for the Calculator, enter the following. DB will be set to the context of the CALC.EXE module. This makes the set of identifiers in the module the currently available ids.

```
>ctx "calc.exe"
```

To see the current lexical scope, enter the ctx command with no parameters.

## Breakpoints: the BP, BL, and BC Commands

Setting a breakpoint at an instruction halts execution when that instruction is hit. Then you can examine the variables and CPU registers to see what your program has been doing up to that point.

With PenPoint's multitasking environment, multiple tasks may simultaneously execute the same code. By default, DB sets breakpoints so that they affect all tasks, but you can restrict a breakpoint so that it only affects a specified set of tasks.

The bp command sets a breakpoint. This example sets a breakpoint at the entry point of CalcEngineEnterOperator:

```
>bp CalcEngineEnterOperator
```

The bl command lists all breakpoints. The listing shows which tasks will be trapped at the breakpoint (in the square brackets), and then shows the location of the breakpoint.

```
>bl
1: [ ] BP CalcEngineEnterOperator.0381
```
After setting a breakpoint, enter the `g` command to resume execution.

```
> g
```

When a breakpoint is hit, DB suspends the task and displays the following information. The first line shows which task hit the breakpoint. The second line displays the source code line on which the breakpoint was set. This is the instruction that is about to be executed.

```
00508 P "go-calculator-v1(0)"[1] 1.272 1: BP
381>> P_CALC_ENGINE_DATA pInst = *pData; // Reduce dereferences.
```

You can also set a breakpoint at a specific line of source code, as follows. If you do not qualify the line number with a file name or a routine name, DB will try to use the file containing the current source line.

```
"go-calculator-v1(0)"[1] 0508>bp @381
"go-calculator-v1(0)"[1] 0508>bp "calceng.c".381
```

A breakpoint can be cleared with the `bc` command.

### Viewing the Call Stack: the ST Command

You can examine a task’s call stack with the `st` command. The call stack maintains local variables and parameters as your program moves from one scope to another.

```
"go-calculator-v1(0)"[1] 0508>st
%1 >
CalcEngineEnterOperator(
  msg: 25170108, self: 01ad103fb, pArgs: 043444c, ctx: 0432be8,
  pData: 043fe93cc)
%2  ObjectCall [ msgCalcEngineEnterOperator [dyn 1ad103fb] 043444c 0 ]
%3  CalcEngineProcessKey(
  msg: 8392892, self: 01ad103fb, pArgs: 043444c, ctx: 0432c48,
  pData: 043fe93cc)
%4  ObjectCall [ msgCalcEngineProcessKey [dyn 1ad103fb] 043444c 043a478d5 ]
%5  CalcAppButtonNotify(
  msg: 117440616, self: 01ac403e5, pArgs: 05, ctx: 0432ca0, pData: 043fe8cb2)
%6  ObjectCall [ msgButtonNotify [dyn lac403e5] 05 01ad90402 ]
%7  ButtonNotifyClient [ 01ad90402 043fe99b0 0432d98 0409bc610 ]
%8  CallMethod [ 07000068 01ad90402 0432d98 0432d60 ]
```

The sequence begins with the top of the stack and then lists the stack back to its base. Repeated `st` commands show more of the stack.

The names of functions are displayed along with the values of the parameters that were passed to them at the time of the call (shown in parentheses). Local variables are listed in the line above the function description, indented and surrounded with curly braces.

The hexadecimal numbers displayed between square brackets (in frames %4, %6, %7, and %8) are the values of variables of functions in modules for which there is no symbolic debugging information.
Frame Numbers and the CTX Command

You can use the frame numbers displayed by the st command as parameters to the ctx command.

```
"go-calculator-vl(0)" [1] 0548> ctx %3
"go-calculator-vl(0)" [1] 0548>ctx
%3 "calc.dll"."calceng.obj".CalcEngineProcessKey."calceng.c" @482
```

You can use the ctx top command to get back to the top of the call stack.

Examining and Setting Values

DB has several commands to examine and set variables, and to evaluate expressions.

The ? (Evaluate) Command

DB's ? ("evaluate") command evaluates a C expression and prints the resulting value.

```
"go-calculator-vl(0)" [1] 0508> pArgs->buf
043444e
```

The ? command can display values in several different formats.

```
"go-calculator-vl(0)" [1] 0508> pArgs->buf
043444e
"go-calculator-vl(0)" [1] 0508> pArgs->buf,s
"6"
"go-calculator-vl(0)" [1] 0508> pArgs->buf,x
043444e
"go-calculator-vl(0)" [1] 0508> pArgs->buf,d
4408398
"go-calculator-vl(0)" [1] 0508> pArgs->buf[0]
54
"go-calculator-vl(0)" [1] 0508> pArgs->buf[0],c
'6'
```

The ? command can be used to structures as well as variables and constants.

```
"go-calculator-vl(0)" [1] 0508> pData
043fe93cc
"go-calculator-vl(0)" [1] 0508> *pData
043474c
"go-calculator-vl(0)" [1] 0508> **pData
{
  xValue: 1.125000; yValue: 77.000000; pendingOp: 5;
  keysSeen: {0, 55, 0, 0, 0, ... 0, 0, 0, 0}; numberEntered: 1;
  calcError: 0; }
```

In this example, DB displays the address for the entry point of a function.

```
>> CalcEngineEnterOperator
043ac7b8
```

The ? command can also set values. Simply put the variable on the left-hand side of a replacement operator (=) and supply a valid expression on the right-hand side. This example changes the first element in the buffer from the value of "6" to the value of "7."
Known Identifiers: the IDS command

The ids command lists the identifiers that are "known" or visible in the current context. These identifiers can be used to set breakpoints, change the value of variables under DB control, and pinpoint locations in the source code that you would like to view.

Identifier Types: the TYPE Command

DB's type command displays the type of its parameter. If the parameter is a structure, type also displays the type of each field.
**Lexical Scope**

The "scope" of identifiers follow the conventions of the C programming language. While in the context of a function, you cannot directly examine the state of variables in another function. Similarly, variables defined within a program block (surrounded by curly braces) are not visible outside of that block.

You can use full Scope.Identifier specification to examine the state of variables that are outside the current DB context. This is discussed in Section 7.2.6.

**Single-Stepping: the p, P, t, and T Commands**

DB provides four commands that allow you to single-step the execution of a program on a statement-by-statement basis:

- `p` Single-steps through source statements, passing over called routines.
- `P` Single-steps through assembly instructions, passing over called routines.
- `t` Single-steps through source statements, stepping into called routines.
- `T` Single-steps through assembly instructions, stepping into called routines.

**Viewing Source Code: the V Command**

The `v` command displays the source code of a program. You can display the source line that is just about to be executed. You can move around in the source files using function declarators, memory addresses, and line numbers to specify which section of the source code to display.

Before viewing source code, you must tell DB which directory contains the source code files. Use either the `srcdir` command or the DBSrc environment variable. (See the description of the `srcdir` command in Chapter 7.)

With the source directory specified, you are ready to display the source code for your program.

This example displays the source code of the beginning of the routine CalcEngineEnterOperator.

```
> v CalcEngineEnterOperator
  378   MsgHandlerWithType(CalcEngineEnterOperator, P_CALC_ENGINE_TOKEN, P_CALC_ENGINE_DATA)
  379   {  
  380     P_CALC_ENGINE_DATA pInst = *pData; // Reduce dereferences
  381     double result;
  382     STATUS s;
  383     if (pArgs->key == nop) {
```

You can also directly specify an instruction memory address to the `v` command. (See Chapter 7 for a description of the `?` command used in this example.)
> CalcEngineEnterOperator 043a1c7b8
>v 043a1c7b8
376 */
377
378 MsgHandlerWithTypes(CalcEngineEnterOperator, P_CALC_ENGINE_TOKEN, \ 379 PP_CALC_ENGINE_DATA)
380 {
381 P_CALC_ENGINE_DATA pInst = *pData; // Reduce dereferences
382 double result;
383 STATUS s;

You can also specify a line number to be viewed. (By default the v command displays lines on either side of the line number requested. See the Datasheet for complete information.)

"go-calculator-v1(0)"[1] 0508>v @204F255D
201 {
202 U16 len;
203 len = strlen(pInst->keysSeen);
204 if (len >= maxDigits-1) {
205 return;
206 }

Whenever DB returns from executing statements, you can display the code that is pending execution with the v and no parameters:

"go-calculator-v1(0)"[1] 0508>v
378 MsgHandlerWithTypes(CalcEngineEnterOperator, P_CALC_ENGINE_TOKEN, \ 379 PP_CALC_ENGINE_DATA)
380 {
377>> P_CALC_ENGINE_DATA pInst = *pData; // Reduce dereferences
381 double result;
382 STATUS s;
383
384 if (pArgs->key == nop) {

Repeating the v command displays the next several lines of source code. In this way you can page down through a source file.

Viewing Assembly Code: the U Command
6.12

The u command displays the assembly code of a program. Variables are displayed as addresses and constants are displayed as the values themselves. The following command displays assembly code starting at the address indicated by the identifier CalcEngineEnterOperator.

>u CalcEngineEnterOperator
43A1C7B8 55 PUSH EBP
43A1C7B9 89 E5 MOV EBP, ESP
43A1C7BB 83 EC 10 SUB ESP, 16
43A1C7BE 8B 45 18 MOV EAX, DWORD PTR [EBP+24{pData}]
43A1C7C1 8B 00 MOV EAX, DWORD PTR [EAX]
43A1C7C3 89 45 FC MOV DWORD PTR [EBP-4{pInst}], EAX
43A1C7C6 8B 45 10 MOV EAX, DWORD PTR [EBP+16{pArgs}]
43A1C7C9 66 83 38 00 CMP WORD PTR [EAX], 0
43A1C7CD 75 05 JNZ 043A1C7D4{CalcEngineEnterOperator.8385}
43A1C7CF E9 AD 01 00 00 JMP 043A1C981{CalcEngineEnterOperator.8426}
If you do not specify an address, DB starts from where it left off in the previous u command.

You can also specify an address directly:

```
>! CalcEngineEnterOperator
043a1c7b8
>u 043a1c7b8
```

```
43A1C7B8 55
43A1C7B9 89 E5
43A1C7BB 83 EC 10
43A1C7BE 8B 45 18
43A1C7C1 8B 00
43A1C7C3 89 45 FC
43A1C7C6 8B 45 10
43A1C7C9 66 83 38 00
43A1C7CD 75 05
43A1C7CF E9 AD 01 00 00
```

```
PUSH EBP
EBP, ESP
EAX, DWORD PTR [EBP+24{pData}]
EAX, DWORD PTR [EAX]
DWORD PTR [EBP-4{plnst}], EAX
EAX, DWORD PTR [EBP+16{pArgs}]
```

```
E9 AD 00 00 00
```

```
CMF CMP WORD PTR [EAX], 0
```

```
043A1C7CD4{CalcEngineEnterOperator.@385}
043A1C981{CalcEngineEnterOperator.@426}
```

**Executing C Code**

DB includes a C language interpreter which can interactively execute code during a debugging session.

With the interpreter, DB can also use C expressions to represent addresses and variables.

When interpreting DB commands, the ! command causes DB to switch to interpreting C code. Conversely, when interpreting C code, the ! command causes the C interpreter to interpret the next statement as a DB command.

**Executing Code at the DB Prompt**

This example executes C code which changes an array of values. The ! command is used to enter C declarations and statements.

```
"go-calculator-v1(0)"[1] 0508>!int i;
"go-calculator-v1(0)"[1] 0508>!for (i=0; i<4; i++) (*pData)->keysSeen[i] = '7';
"go-calculator-v1(0)"[1] 0508>! (*pData)->keysSeen, s
"7777"
```

**Executing Code at a Breakpoint**

You can specify code to be executed when a breakpoint is hit.

```
>bp CalcEngineEvalBinary {
    printf("op=%d opnd2=%f\n", op, opnd2);
    if ((op==5) && (opnd2==0.0)) {
        printf("got divide by zero!\n");
        !break;
    }
}
```

```
>g
```

Notice that the break command in the if-statement is prefixed with the ! command; this tells the C interpreter that the following command is a DB command rather than C code.
Also notice that while we’re entering C code into a block delimited by braces ({{ and }}), DB’s prompt changes from a greater-than sign (>) to a brace (}).

Now when the Calculator’s buttons are tapped, the breakpoint is hit and DB executes the code inside the curly braces of the bp command. The printf statement prints out the current value for some variables. The break command returns control to the DB prompt.

\[
\begin{align*}
op &= 5 \ \text{opnd2} = 9.000000 \\
op &= 2 \ \text{opnd2} = 6.000000 \\
op &= 4 \ \text{opnd2} = 3.000000 \\
op &= 3 \ \text{opnd2} = 9.000000 \\
op &= 5 \ \text{opnd2} = .000000 \\
\text{got divide by zero!}
\end{align*}
\]

...  

**Task List: the TL Command**  

To see the list of all tasks, issue this command:

```
>tl  
```

```
00508 P "go-calculator-v1(0)"[1] 3.199 1: BP  
00508 P "go-calculator-v1(0)"[1] 3.199 1: BP  
80>> STATUS s = stsOK;  
```

The process names that are followed by a zero in square brackets ([0]) are the application monitor processes for the installed application. The process names that are followed by numbers greater than zero in square brackets (such as [1] and [2]) are the processes associated with active documents.

**Saving Typing**  

DB provides several ways to save typing common commands.

**DB.INI File**  

At system start-up time, DB looks for a startup file; if it finds one, it reads commands from the file. You can use the startup file to store a startup sequence for your debugging session, accelerating the debugging part of a edit-compile-debug cycle.
DB first looks for the DBINI environment variable in \PENPOINT\BOOT\ENVIRON.INI:

    DBINI=\c\mydir\mydb.ini

If DBINI is not defined, DB looks for \PENPOINT\BOOT\DB.INI and runs it if it exists. If DB doesn't find a DB.INI, it returns control to PenPoint (it acts as if a user typed a "g" command).

Using DB Scripts

You can use the < command to read a file that contains DB commands. For example, you could create a DOS file \GO\CALC\SCRIPT.DB that contains the following lines:

    sym "calculator"
    sym "calc eng"
    srcdir "calc_eng" \c\penpoint\sdk\sample\calc
    srcdir "calculator" \c\penpoint\sdk\sample\calc

Then all you have to do is enter the following command when you want to link the symbolic debugging information into your DB session:

    > < \c\go\calc\script.db

Command Line Editing

You can edit DB command lines with these keys: [c], [d], [Home], [End], [Del], and [Backsp]. You can use the [Esc] key to cancel the entire command. The “cursor” is shown as a tiny dot.

The up and down arrow keys recall previous commands, which can be edited, if necessary.

DB and Memory Use

The symbolic debugging information used by DB consumes memory. Combined with the overhead for DB itself, your program may run into memory constraints.

Checking Available Memory: the MI Command

You can check on available memory with DB's mi command.

Eliminating Applications

To conserve memory, you can boot a reduced PenPoint configuration by removing unneeded applications from \PENPOINT\BOOT\APP.INI. Once PenPoint is running, you can delete documents and de-install or deactivate applications in the usual manner.
Loading Partial Symbolic Debugging Information

Using the compiler's /D2 switch and the linker's DEBUG ALL switch produces a lot of debugging information; DB may not have enough memory to read it all. To get around this, the sym command has an optional form.

The following command will load full symbolic debugging information for obj1.obj and obj2.obj, but only line numbers and global identifiers for the rest of the module's object files.

```bash
> sym "myapp" (obj1, obj2)
```

The following command, with no files listed between the parentheses, loads only line numbers and global identifiers for all object files in the module:

```bash
> sym "myapp" ()
```

See the Datasheet for the sym command for details.

Using DB to Send Messages

You can ask DB to have current task execute ObjectCall, ObjectCallAncestor, ObjectSend, ObjectSendUpdate, ObjectPost or ObjectPostAsync by using those functions in a DB ? command. For instance:

```bash
> ?ObjectCall(msg, object, 0)
```

String Names for Messages, Objects, and Statuses

You can "encode" a PenPoint message, object/class, or status name within any DB expression with the construct

```
[msg|obj|cls <name>]
```

For instance, the following will send msgDump to theProcess.

```bash
> ?ObjectCall([msg msgDump], [obj theProcess], 0)
```

If you have loaded the class manager symbols, DB can interpret PenPoint message, object, class, and status identifiers. Thus you could enter the previous command as:

```bash
> ?ObjectCall(msgDump, theProcess, 0)
```

Watching Memory: the ON ACCESS and ON STORE Commands

You can use DB's on command to interrupt execution when memory locations are read or written. There are two variations that are useful for watching memory:

- **on store lvalue** This variation breaks to DB when the memory specified by lvalue is written.
- **on access lvalue** This variation breaks to DB when the memory specified by lvalue is read or written.
An lvalue is typically a variable name, although it can also be any expression referencing a region of storage. For a complete definition of lvalues, see any good reference book on the C programming language, such as *The C Programming Language* by Kernighan and Ritchie.

In the following example, assume that you have taken a breakpoint at the routine CalcEngineEnterOperator and that you decide that you need to break whenever the value of (*pData)->numberEntered changed.

```
"go-calculator-v1(0)"[1] 0508>**pData
{
  xValue: 3.092736e-14; yValue: 3.092736e-14; pendingOp: 5;
  keysSeen: {0, 55, 0, 55, 56,... 0, 0, 0, 0}; numberEntered: 0;
  calcError: 0;}
"go-calculator-v1(0)"[1] 0508>on store (*pData)->numberEntered
"go-calculator-v1(0)"[1] 0508>b1
 1: [*] on Store [target task 0508 logAddr 43477C] 1 2 { }
```

Execution will halt immediately after (*pData)->numberEntered has been written; you may have to give the `v b` command to display the previous source code.

```
00508 P "go-calculator-v1(0)"[1] 1.948 1: Store
430>> return stsOK;
"go-calculator-v1(0)"[1] 0508>v
427
428  }
429
430>> return stsOK;
431  MsgHandlerParametersNoWarning;
432
433 } /* CalcEngineEnterOperator */
434
"go-calculator-v1(0)"[1] 0508>v b
419  Dbg(if (s != stsCalcEngineComputeError) StsWarn(s));
420  return s;
421
422  pInst->pendingOp = nop;
423  CalcEngineSaveValue(result, pArgs, pInst);
424
425  pInst->pendingOp = pArgs->key;
426  pInst->numberEntered = false;
"go-calculator-v1(0)"[1] 0508>g
00508 P "go-calculator-v1(0)"[1] 2.106 1: Store
366>> return stsOK;
"go-calculator-v1(0)"[1] 0508>v
363  CalcEngineSaveValue(pArgs->value, pArgs, *pData);
364  (*pData)->numberEntered = true;
365
366>> return stsOK;
367  MsgHandlerParametersNoWarning;
368
369 } /* CalcEngineEnterNumber */
370```
If you want to break whenever memory is either read or written, use `on access`.

```
"go-calculator-v1(0)"[1] 0508> on access (*pData)->numberEntered
"go-calculator-v1(0)"[1] 0508> bl
1: [*] on Access [target task 0508 logAddr 43477C] 1 2 { }
```

Now execution will halt whenever the location is read or written. For example, notice that the calculator application now stops at line 416:

```
"go-calculator-v1(0)"[1] 0508> g
00508 P "go-calculator-v1(0)"[1] 2.302 1: Access
416>> if ((pInst->pendingOp != nop) AND (pInst->numberEntered)) { 
```

These two commands can operate on raw memory addresses as well as on variables. As examples, you may want to use a raw address if you need to watch allocated memory or if you're debugging a module without symbolic debugging information. In the following example, we'll compute the address to watch.

```
"go-calculator-v1(0)"[1] 0508> ? &((*pData)->numberEntered)
043477c
"go-calculator-v1(0)"[1] 0508> type (*pData)->numberEntered
short
"go-calculator-v1(0)"[1] 0508> on access *(short *)043477c
```

### Events that Activate DB

There are three circumstances which lead to the DB prompt appearing:

- If the user types DB's Pause key.
- If any task experiences a fault, such as a page fault or divide by zero.
- If any code calls the Debugger PenPoint function.

### Exiting DB and PenPoint

To exit DB and PenPoint, enter the `q` command at the DB prompt:

```
> q
```
Chapter 7 / DB Command Reference

This chapter describes the commands that you can enter at the DB prompt. Most of the command descriptions are accompanied by examples of command usage. Be aware that the line numbers that appear in these examples might not match line numbers that you see while examining the same sample application.

Command Summary

Table 7-1 lists many of DB's commands in functional categories.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
<th>Process and Task Commands</th>
<th>Execution Control Commands</th>
<th>Breakpoint Commands</th>
<th>Display Commands</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctx</td>
<td>Selects the current task and sets DB's lexical scope.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>tl</td>
<td>Display PenPoint's task list.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ti</td>
<td>Display information about a task.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>g</td>
<td>Resumes PenPoint execution.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P</td>
<td>Single-steps assembly instructions, passing over called routines.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>p</td>
<td>Single-steps source statements, passing over called routines.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>Single-steps assembly instructions, stepping into called routines.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>t</td>
<td>Single-steps source statements, stepping into called routines.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q</td>
<td>Quits DB.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>bp</td>
<td>Sets a breakpoint.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>bl</td>
<td>Lists breakpoints.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>bc</td>
<td>Clears the specified breakpoint.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>on</td>
<td>Creates a &quot;handler&quot; for specified events.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>bd</td>
<td>Disables the specified breakpoint.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>be</td>
<td>Enables the specified breakpoint.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>!</td>
<td>Switches the interpreter from DB commands to C code, or vice-versa.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>?</td>
<td>Evaluates and/or assigns a C expression.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>d (db, dw, dd)</td>
<td>Displays memory contents.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>u</td>
<td>Displays assembly code.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>uv</td>
<td>Displays assembly code with interspersed source code.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

continued
Table 7-1 (continued)

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>v</td>
<td>Displays source code.</td>
</tr>
<tr>
<td>vu</td>
<td>Displays source code with interspersed assembly language statements.</td>
</tr>
<tr>
<td>st</td>
<td>Display the call stack.</td>
</tr>
<tr>
<td>r</td>
<td>Displays the registers.</td>
</tr>
<tr>
<td>ai</td>
<td>Displays address information.</td>
</tr>
<tr>
<td>ids</td>
<td>Displays the identifiers known in a scope.</td>
</tr>
<tr>
<td>fns</td>
<td>Displays the functions known in a scope.</td>
</tr>
<tr>
<td>vars</td>
<td>Displays variables known in a scope.</td>
</tr>
<tr>
<td>id</td>
<td>Displays the type and declaration information for an identifier.</td>
</tr>
<tr>
<td>type</td>
<td>Displays type that a C expression evaluates to.</td>
</tr>
<tr>
<td>cm, co, cs</td>
<td>Converts messages, objects, and statuses into their string formats.</td>
</tr>
</tbody>
</table>

File Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>srcdir</td>
<td>Tells DB where to find source files for a module.</td>
</tr>
<tr>
<td>sym</td>
<td>Loads symbolic debugging information into memory.</td>
</tr>
<tr>
<td>&lt;</td>
<td>Reads a file as input for DB.</td>
</tr>
<tr>
<td>files</td>
<td>Displays the files associated with a scope.</td>
</tr>
</tbody>
</table>

Profiling Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>profile</td>
<td>Creates a profile breakpoint.</td>
</tr>
<tr>
<td>dp</td>
<td>Displays profile data.</td>
</tr>
<tr>
<td>zp</td>
<td>Clears profile data.</td>
</tr>
</tbody>
</table>

Miscellaneous Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fl</td>
<td>Lists values of debugging flags.</td>
</tr>
<tr>
<td>fs</td>
<td>Sets values of debugging flags.</td>
</tr>
<tr>
<td>ver</td>
<td>Displays DB's version.</td>
</tr>
<tr>
<td>h</td>
<td>Displays help on DB commands and topics.</td>
</tr>
<tr>
<td>log</td>
<td>Starts and stops logging.</td>
</tr>
<tr>
<td>mi</td>
<td>Displays memory information.</td>
</tr>
<tr>
<td>mini</td>
<td>Enters PenPoint's mini-debugger.</td>
</tr>
<tr>
<td>od</td>
<td>Sends msgDump to an object.</td>
</tr>
<tr>
<td>break</td>
<td>Returns control to the DB prompt.</td>
</tr>
</tbody>
</table>

Notation Conventions

This chapter uses the notation conventions described in the next few sections.

Scope Specification (scopeSpec)

A scope specification (or scopeSpec) specifies a position in a task's stack. Changing the scope shifts the lexical scope for DB within the task. When examining variables, you will see values for the current scope.
A scopeSpec takes the following forms:

\[
\begin{array}{ll}
  [ \{ \text{into} \mid \text{outto} \} \} \{ \text{functionName} & \\
  & \text{filename} & \\
  & \text{programName} & \\
  & \text{scopeName} & \\
  & @ \text{lineNumber} & \\
  & \% \text{stackFrameNumber} \}
\end{array}
\]

or a string of the following elements separated by periods (to cause multiple moves into or out of a scope level):

\[
\text{top} \mid \text{in} \mid \text{out} \mid \text{inb} \mid \text{outb}
\]

### Table 7-2
Scope Specification

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>into</td>
<td>Causes DB to move in to the scope of the function or line number after the dot.</td>
</tr>
<tr>
<td>outto</td>
<td>Causes DB to move out to the scope of the function or line number after the dot.</td>
</tr>
<tr>
<td>functionName</td>
<td>A function available in the current scope.</td>
</tr>
<tr>
<td>filename</td>
<td>A file name established with the srcdir and v commands, or the name of a loaded module (with file extension).</td>
</tr>
<tr>
<td>programName</td>
<td>The name of a load module. (These names are listed by the sym command.)</td>
</tr>
<tr>
<td>scopeName</td>
<td>The name of a scope.</td>
</tr>
<tr>
<td>%stack frame number</td>
<td>The number for a scope shown by the st command.</td>
</tr>
<tr>
<td>@lineNumber</td>
<td>A line number in the source file associated with the current scope. The “@” is required. (The v command shows the assignment of line numbers to source files.)</td>
</tr>
<tr>
<td>top</td>
<td>Positions DB at the top of a task's call stack (the innermost scope).</td>
</tr>
<tr>
<td>in</td>
<td>Move “in” one CALL on the stack.</td>
</tr>
<tr>
<td>out</td>
<td>Move “out” one CALL on the stack.</td>
</tr>
<tr>
<td>inb</td>
<td>Move in one lexical block.</td>
</tr>
<tr>
<td>outb</td>
<td>Move out one lexical block.</td>
</tr>
</tbody>
</table>

For example, a scopeSpec is supplied to the ctx command (described elsewhere).

The following commands change the scope of the current task:

```
"go-calculator-v1(0)"[1] 0508>ctx outto.CalcEngineProcessKey
"go-calculator-v1(0)"[1] 0508>ctx
%4 "calc.dll"."calceng.obj".CalcEngineProcessKey."calceng.c".@478
"go-calculator-v1(0)"[1] 0508>ctx in
"go-calculator-v1(0)"[1] 0508>ctx
%3 go-kernel3-V0(1).01:06bea
"go-calculator-v1(0)"[1] 0508>ctx out
"go-calculator-v1(0)"[1] 0508>ctx
%4 "calc.dll"."calceng.obj".CalcEngineProcessKey."calceng.c".@478
"go-calculator-v1(0)"[1] 0508>ctx top
"go-calculator-v1(0)"[1] 0508>ctx
%1 "calc.dll"."calceng.obj".CalcEngineEvalBinary."calceng.c".@76
```
Line Count (*lineCount*)

Several DB commands (for instance, v and u) take an optional parameter known as a *lineCount*. The line count controls how many lines are displayed, and where the display of lines starts.

\[ \text{[ } a \mid b \mid 1 \text{ ] [ } \text{count} \text{]} \]

- **count** The number of lines to be displayed.
- **a** "Around." The command displays \( \frac{1}{2} \) of count lines before the target and \( \frac{1}{2} \) after.
- **b** "Before." The command displays count lines up to and including the target.
- **l** "List." The command displays count lines starting at the target.

Code Addresses

A *codeAddress* is:

- A C function identifier.
- A line number.
- A literal address specification.
- Any C expression that yields a function pointer.

For example, the g command can optionally set a temporary breakpoint at the code address of a routine and then let execution proceed up to that point:

```
>g CalcEngineEvalBinary
```

Data Addresses

A *dataAddress* is:

- A C pointer variable identifier.
- A literal address specification.
- A C expression that yields a pointer.

Line Numbers

A line number is expressed by the @ sign followed by a decimal literal. If the line number is not qualified, DB takes the line to be in the same file as the current source position.

You can qualify the line number with a *scopeSpec*, for example "calc.c".@104 or SomeRoutine.@104.

"Scope.Identifier" Referencing

An identifier outside of the current scope can be referenced with a *Scope.Identifier* reference. The reference consists of a *scopeSpec* followed by a period followed by an identifier. For example:
references the function CalcEngineEvalBinary as it occurs in the scope of its load module "calc_eng." This reference can occur anywhere that a standard C identifier can.

**Task Set (taskSet)**

Many DB commands operate on a set of tasks, or *taskSet*. A task set is one or more task IDs. If necessary, a task set can be syntactically distinguished from a command by enclosing the task set in square brackets.

There are several ways to specify a task set; most DB users will only need the first two or three forms.

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>taskId</td>
<td>a single task</td>
</tr>
<tr>
<td>*</td>
<td>all tasks</td>
</tr>
<tr>
<td>.</td>
<td>the current task</td>
</tr>
<tr>
<td>taskId *</td>
<td>all the tasks in the specified task's parent process</td>
</tr>
<tr>
<td>taskSet +/- taskSet</td>
<td>union or difference of two taskSets</td>
</tr>
<tr>
<td>&quot;progName&quot;</td>
<td>the program name bound to a .EXE or .DLL module, as listed by the tl command</td>
</tr>
<tr>
<td>&quot;progName&quot;[instanceNumber]</td>
<td>an instance of an .EXE as listed by tl</td>
</tr>
<tr>
<td>&quot;#taskName&quot;</td>
<td>a task name, as set by the OSTaskNameSet() function</td>
</tr>
</tbody>
</table>

**Command Datasheets**

This section contains a complete reference data sheet for each of the DB commands. Each datasheet explains all the arguments in a command, and examples of usage are given. The commands are listed alphabetically.

### !

Switches the interpreter from DB commands to C code, or vice-versa.

**Syntax**

```c
!  C code
```

**C code** Any legal C declaration or statement.

**Examples**

```c
> !typedef int LENGTH;
> !LENGTH maxlen;
> !maxlen = 2;
```

**Remarks**

The ! operator can be used to declare new data.

The ! operator can also be used to escape to a DB command where DB is expecting a C statement. --Refer to Section 6.13.
Part 2 / Debugging Penpoint Applications

Reads a file as input for DB. The file should contain DB commands or C code.

Syntax

\(<\ filename\>

\textit{filename} \quad \text{A PenPoint file name.}

Examples

\(\text{> < script.db}\)

Remarks

If the filename ends in ".c" the file is treated as C code; otherwise it is treated as a collection of DB commands.

?\n
Evaluates a C expression and prints the resulting value. It can also assign a value to a variable.

Syntax

\(?\ C-expression\ [\ \textit{fmt}\ ]\)

\(?\ C-expression = C-expression\)

\textit{C-expression} \quad \text{Any expression that can be evaluated within the current context. Identifiers that are outside the current scope can be used, but must be qualified as described in Section 7.2.6. Identifiers from another task cannot be referenced. Identifiers from an inner scope cannot be referenced.}

\textit{fmt} \quad \text{Optionally tells DB how to format the value. When \textit{fmt} is used, the comma is required. If the value of the expression is a struct, a string of format characters may be specified, one for each element. These formats are allowed:}

- \textit{d} \quad \text{displays a decimal value}
- \textit{x} \quad \text{displays a hex value}
- \textit{p} \quad \text{displays a pointer}
- \textit{s} \quad \text{displays a string}
- \textit{c} \quad \text{displays a character}

Examples

\"go-calculator-v1(0)\"[1] 0508>?\texttt{pArgs->buf}
043444e
\"go-calculator-v1(0)\"[1] 0508>\texttt{pArgs->buf, s}
\"6\"
\"go-calculator-v1(0)\"[1] 0508>\texttt{pArgs->buf, x}
043444e
\"go-calculator-v1(0)\"[1] 0508>\texttt{pArgs->buf, d}
4408398
\"go-calculator-v1(0)\"[1] 0508>\texttt{pArgs->buf, s}
\"6\"
\"go-calculator-v1(0)\"[1] 0508>\texttt{pArgs->buf[0]}
54
\"go-calculator-v1(0)\"[1] 0508>\texttt{pArgs->buf[0], c}
\'6\'
"go-calculator-v1(0)"[1] 0508>?pData
043fe93cc
"go-calculator-v1(0)"[1] 0508>?*pData
043474c
"go-calculator-v1(0)"[1] 0508>?**pData
{
  xValue: 1.125000; yValue: 77.000000; pendingOp: 5;
  keysSeen: {0, 55, 0, 0, 0, ... 0, 0, 0, 0}; numberEntered: 1;
  calcError: 0;
}
"go-calculator-v1(0)"[1] 0508>? CalcEngineEnterOperator
043a1c7b8
"go-calculator-v1(0)"[1] 0508>?pArgs->buf[0],c
'6'
"go-calculator-v1(0)"[1] 0508>?pArgs->buf[0] = '7'
55
"go-calculator-v1(0)"[1] 0508>?pArgs->buf[0],c
'7'

Remarks
If the value being printed by ? is a structure, fmt can be a string to specify formats for the structure members.

ai
Displays address information.

Syntax

ai dataAddress

dataAddress A data address as described in Section 7.2.4.

Examples

"go-calculator-v1(0)"[1] 0508>ai pArgs
task 0508 000434000.000437fff user private rw offset 044c

bgNc
Clears the specified breakpoint.

Syntax

bc { breakpoint | * }

breakpoint DB's breakpoint number as displayed by the bl command.

* all breakpoints.

Examples

Clears breakpoint 2.
>bc 2

Clears all breakpoints.
>bc *

Remarks
The bc command permanently clears a breakpoint; see the bd command for a discussion of how to temporarily disable breakpoints.
**bd**
Temporarily disables the specified breakpoint.

**Syntax**
```
bd { breakpoint | * }
```
- `breakpoint` DB's breakpoint number as displayed by the `bl` command.
- `*` all breakpoints.

**Examples**
In the following example, notice how the disabled breakpoint is displayed by the `bl` command.
```
>bp CalcEngineEvalBinary
>bp CalcEngineEnterNumber
>bd 1
>bl
1: [*] BP (disabled) CalcEngineEvalBinary.@76
2: [*] BP CalcEngineEnterNumber.@359
```

**Remarks**
The `bd` command temporarily disables one or all breakpoints. This is particularly useful if you've set up complex breakpoints that you'd like to temporarily disable without having to re-enter them again later.

If `breakpoint` is a profile breakpoint, `bd` stops accumulating execution data, but retains its current data. See the `zp` command for information about clearing data.

See the `bc` command for a discussion of how to permanently remove breakpoints.
See the `be` command for a discussion of how to re-enable breakpoints.

---

**be**
Enables the specified breakpoint.

**Syntax**
```
be { breakpoint | * }
```
- `breakpoint` DB's breakpoint number as displayed by the `bl` command.
- `*` all breakpoints.

**Remarks**
See the `bd` command for a discussion of how to disable breakpoints.

---

**bl**
Lists breakpoints.

**Syntax**
```
bl
```

**Examples**
```
>bp CalcEngineEvalBinary
> bp CalcEngineEnterNumber
>profile +c CalcEngineEvalUnary
>bl
1: [*] BP CalcEngineEvalBinary.@80
2: [0508] BP CalcEngineEnterNumber.@363
3: [*] Profile +c go-calc_eng-V1(0) 1.2aa:call
```

**Remarks**
The `bl` command lists breakpoints set with the `bp` or `profile` commands.

Technically, `bl` lists all event handlers. Breakpoints and profiles are by far the most common event handlers. See the discussions of the `on`, `on access` and `on store` commands for descriptions of other event handlers.
**bp**

Sets a breakpoint.

Syntax

\[
\text{[taskSet]} \text{ bp codeAddress [eventAction]} \]

- **taskSet**: A task set as described in Section 7.2.7. Defaults to all tasks.
- **codeAddress**: The address for the breakpoint as described in Section 7.2.3.
- **eventAction**: C code to be executed when the breakpoint is hit.

Examples

This command sets a breakpoint at the entry point of a function for all tasks.

>bp CalcEngineEvalBinary

Sets a breakpoint at the entry point of a function in a single task.

>0508 bp CalcEngineEvalBinary

Sets a breakpoint at the instruction at specific code address for all tasks.

>bp 043a1c195

Sets a breakpoint at line 377 of the current source context.

>bp @377

Sets a breakpoint at the entry to CalcEngineEvalBinary. Whenever this breakpoint is hit, the contents of the variables `op` and `opnd2` are printed. Under the right conditions, control is returned to DB; otherwise the program simply continues execution.

>bp CalcEngineEvalBinary {
  printf("op=%d opnd2=%f\n", op, opnd2);
  if ((op==5) && (opnd2==0.0)) {
    printf("got divide by zero!\n");
    !break;
  }
}
**CM, CO, CS**

Converts messages, objects, and statuses into their string formats.

**Syntax**

```
CM expression
CO expression
CS expression
```

*expression* An expression (of the appropriate type) to be converted.

**Examples**

```
>cs 0
stsOK
>cs 081000002
stsBadObject
```

**CTX**

Sets DB's context. Sets the current task and sets the lexical scope. When no parameters are specified, ctx displays the current context.

**Syntax**

```
[taskID] ctx [scopeSpec]
```

*taskID* A task ID as described in Section 7.2.7. If no *taskSet* is specified, the current task is operated upon. The set of current task IDs is available via the tl command.

*scopeSpec* A scopeSpec as defined in Section 7.2.1. Defaults to the top of the task's call stack.

**Examples**

All of the following examples assume that task 0508 has the following stack trace:

"go-calculator-vl(0)"[1] 0508>st

```
%1 >
CalcEngineEvalBinary(
%2
CalcEngineEnterOperator(
  msg: 25170108, self: 01ad203fb, pArgs: 043444c, xxx: 0432be8,
  pData: 043ff0608)
%3 ObjectCall [ msgCalcEngineEnterOperator [dyn lad203fb] 043444c 0 ]
%4
CalcEngineProcessKey(
  msg: 8392892, self: 01ad203fb, pArgs: 043444c, xxx: 0432c48,
  pData: 043ff0608)
%5 ObjectCall [ msgCalcEngineProcessKey [dyn 1ad203fb] 043444c 043a4c8d5 ]
%6
CalcAppButtonNotify(
  msg: 117440616, self: 01ac503e5, pArgs: 05, xxx: 0432ca0, pData: 043fefe)
%7 ObjectCall [ msgButtonNotify [dyn lac503e5] 05 01ada0402 ]
%8 ButtonNotifyClient [ 01ada0402 043ff0bec 0432d98 0409bc610 ]
```

Sets DB’s context to a specific task. Note that DB’s prompt changes.

```
>0508 ctx
"go-calculator-vl(0)"[1] 0508>
```
Displays DB’s current context.

```
"go-calculator-v1(0)"[1] 0508>ctx
%1  "calc.dll"."calceng.obj".CalcEngineEvalBinary."calceng.c".@80
```

Causes DB to move out to the scope of CalcEngineEnterOperator.

```
"go-calculator-v1(0)"[1] 0508>ctx outto.CalcEngineEnterOperator
"go-calculator-v1(0)"[1] 0508>ctx
%2  "calc.dll"."calceng.obj".CalcEngineEnterOperator."calceng.c".@417
```

The following example move’s DB context to the top of the stack and then out two stack frames. The context is then moved in one stack frame.

```
"go-calculator-v1(0)"[1] 0508>ctx top
"go-calculator-v1(0)"[1] 0508>ctx out.out
"go-calculator-v1(0)"[1] 0508>ctx
%3 go-kernel3-V0(1).01: 06bea
"go-calculator-v1(0)"[1] 0508>ctx in
"go-calculator-v1(0)"[1] 0508>ctx
%2  "calc.dll"."calceng.obj".CalcEngineEnterOperator."calceng.c".@417
```

The following move the context out the stack to the frame corresponding to CalcAppButtonNotify.

```
"go-calculator-v1(0)"[1] 0508>ctx outto.CalcAppButtonNotify
"go-calculator-v1(0)"[1] 0508>ctx
%6  "calc.exe"."calcapp.obj".CalcAppButtonNotify."calcapp.c".@324
```

Causes DB to switch to the fourth stack frame.

```
"go-calculator-v1(0)"[1] 0548>ctx %4
"go-calculator-v1(0)"[1] 0548>ctx
%4  "calc.dll"."calceng.obj".CalcEngineProcessKey."calceng.c".@482
```

Causes DB to switch to the task 0508, and position to the top of that task’s call stack.

```
> 0508ctx top
```

This variation of the context command makes the set of ids defined in the “calculator” module be the set of ids currently visible. Notice that the prompt does not change. (See the Datasheet for the ids command for more information.)

```
>ctx "calculator"
>ids
{(tags) APP_NEW APP_NEW_ONLY APP_OPEN CALC_APP_DATA CALC_ENGINE_TOKEN
CLS_SYM.MSG CLS_SYM.OBJ CLS_SYM_STS FS_LOCATOR OBJ_RESTORE OBJ_SAVE
OBJECT_NEW (vars) calcMsgSymbols calcObjSymbols calcStsSymbols
clsCalcAppTable (functions) CalcAppInit CalcAppButtonNotify CalcAppClose
CalcAppCreateButtons CalcAppCreateCalcWindow CalcAppCreateDisplayWindow
CalcAppDisplayString CalcAppFree CalcAppInit CalcAppLayoutButtons CalcAppOpen
CalcAppRestore CalcAppSave CalcSymbolsInit clsCalcAppInit main}
```

Remarks

When changing tasks, the `ctx` command changes DB’s prompt so that it reflects the new task.
Part 2 / Debugging Penpoint Applications

**Syntax**

\[
d \quad [b \mid w \mid d] \ [dataAddress] \ [length]
\]

- **d**: Display memory contents. Without one of the following options, defaults to the display format of the previous d, db, dw, or dd command.
- **db**: Display memory contents as bytes.
- **dw**: Display memory contents as 2-byte words, which are displayed as four hexadecimal digits. DB assumes the Intel reversed-word ordering—the most significant byte of the word comes second in address space.
- **dd**: Display memory contents as 4-byte double-words, which are displayed as eight hexadecimal digits. DB assumes the Intel reversed-word ordering for double-words and within 16-bit words—the most significant word in the double-word comes second in address space.

- **dataAddress**: Start address for the display, as described in Section 7.2.4. If omitted, DB starts from where it left off in the previous d, db, dw, or dd command.

- **length**: Number of bytes to display.

**Examples**

```
"go-calculator-v1(0)"[1] 0508>db pArgs
434440 - 05 00 31 00 1.
434450 30 30 30 30 30 30 30 30 30 30 AC 07 00 D8 05 04 A8 00000000,..X..(  
434460 AC 07 00 CD 05 04 48 5F-07 00 C3 05 00 00 00 ,..M...H...C.....  
434470 00 C0 58 40 FB 03 D2 1A-FC 03 D4 1A FD 03 D5 1A .@X@(.R.].T..).U.  
434480 FE 03 D6 1A AE AE 08 05-72 FA 02 E0 ~.V.....zz.
"go-calculator-v1(0)"[1] 0508>dw pArgs
434440 0005 0031  
434450 3030 3030 3030 AC 03 0007 05D8 A004  
434460 07AC CD00 0405 5F48 0007 05C3 0000 0000  
434470 C000 4058 03FB 1AD2 03FC 1AD4 03FD 1AD5  
434480 03FE 1AD6 AEAE 0508 FA72 E002  
"go-calculator-v1(0)"[1] 0508>dd pArgs  
434440  
434450 30303030 30303030 007AC30 A0405D8  
434460 CD0007AC 5F4B0405 05C30007 00000000  
434470 4058C000 1AD203F8 1AD403FC 1AD503FD  
434480 1AD603FE 0508AEAE E002FA72  
"go-calculator-v1(0)"[1] 0508>d pArgs  
434440 00310005  
434450 30303030 30303030 007AC30 A0405D8  
434460 CD0007AC 5F4B0405 05C30007 00000000  
434470 4058C000 1AD203F8 1AD403FC 1AD503FD  
434480 1AD603FE 0508AEAE E002FA72
```

**dp**

Displays profile data.

**Remarks**

See Chapter 8, which discusses profiling and the dp command in detail.
files
Display the files associated with a scope.

Syntax
files [scopeSpec]

scopeSpec  A scopeSpec as defined in Section 7.2.1. If not specified, files displays all of the known files.

Examples
>srcdir "calc.dll" \c\penpoint\sdk\sample\calc
>srcdir "calc.exe" \c\penpoint\sdk\sample\calc
>files
"calc.dll"
"calc.dll"."cengmeth.obj" (obj)
"calc.dll"."calceng.obj" (obj)
"calc.dll"."calceng.obj"."calceng.c" (src) in \c\penpoint\sdk\sample\calc
"calc.dll"."apprefs.obj" (obj)
"calc.exe"
"calc.exe"."cappmeth.obj" (obj)
"calc.exe"."calcapp.obj" (obj)
"calc.exe"."calcapp.obj"."calcapp.c" (src) in \c\penpoint\sdk\sample\calc
"calc.exe"."s_calc.obj" (obj)
"calc.exe"."s_calc.obj"."s_calc.c" (src) in \c\penpoint\sdk\sample\calc
"calc.exe"."appstart.obj" (obj)

Remarks
This command displays a subset of the information displayed by the ids command.

fl
Lists values of debugging flags.

Syntax
fl [flagSet]

flagSet  Either a single printing character or two hex digits. If missing, fl displays the value of all debugging flags.

Examples
>fl
42 B 00000769  44 D 00008000  47 G 00000040  5a Z 08000000

Remarks
Only those flags which contain at least one non-zero value are displayed.
See the Datasheet for fs for a discussion of setting debugging flags.
fns

Display the functions known in a scope.

Syntax

fns [scopeSpec]

scopeSpec A scopeSpec as defined in Section 7.2.1. Defaults to showing the functions in the current scope and each of its parent scopes.

Examples

"go-calculator-v1(0)"[1] 0508>fns
{}
{}
{}
{}
{ (functions) CalcEngineAppendCharToToken CalcEngineDump CalcEngineEnterNumber CalcEngineEnterOperator CalcEngineEvalBinary CalcEngineEvalUnary CalcEngineFree CalcEngineInit CalcEngineProcessKey CalcEngineRestore CalcEngineSave CalcEngineSaveValue CalcEngineUpdateToken DllMain}

"go-calculator-v1(0)"[1] 0508>fns "calc_eng"
{ (functions) CalcEngineAppendCharToToken CalcEngineDump CalcEngineEnterNumber CalcEngineEnterOperator CalcEngineEvalBinary CalcEngineEvalUnary CalcEngineFree CalcEngineInit CalcEngineProcessKey CalcEngineRestore CalcEngineSave CalcEngineSaveValue CalcEngineUpdateToken DllMain}

Remarks

This command displays a subset of the information displayed by the ids command.

fs

Sets values of debugging flags.

Syntax

fs flagSet [ [ + | ] flagValue ]

flagSet Either a single printing character or two hex digits.

flagValue A hex value. If not present, fs displays the current value of flagSet.

+ If present, turn the bits in flagValue on.

– If present, turn the bits in flagValue off.

Examples

>fl G
040
>fs G 44
>fl G
044
>fl G + 800
>fl G
0844
>fs G 40
>fl G
040

Remarks

See the Datasheet for fl for a discussion of listing the values of debugging flags.
**g**
Resume PenPoint execution.

**Syntax**

\[ \text{[taskSet]} \ g \ [\text{codeAddress}] \]

- **taskSet** A task set as defined in Section 7.2.7. Defaults to the current task.
- **codeAddress** If specified, a temporary breakpoint is set at the code address. Execution continues until this or any other breakpoint is encountered. The address is specified as described in Section 7.2.3.

**Examples**

Starts execution of the current task after setting a breakpoint at the entry point for CalcEvalBinary.

>g CalcEngineEvalBinary

**Remarks**

Execution continues until PenPoint detects a fault condition, or DB detects a breakpoint, or a task in taskSet terminates.

Only tasks with non-zero freeze counts will run.

**h**
Display help on DB commands and topics.

**Syntax**

**h** *topic*

**Examples**

Type the following to get a list of available topics:

>h

Type the following to get a list of DB commands:

>h commands

**id**
Display type and declaration information for an identifier.

**Syntax**

\[ \text{id} \ { \text{identifier} \ | \ \text{address} } \]

- **identifier** A function or variable identifier from your program.
- **address** DB will try to find an identifier near this address.

**Examples**

"go-calculator-v1(0)"[1] 0508>id pInst
program local pInst: ptr to struct CALC_ENGINE_DATA @ [bp-16]

"go-calculator-v1(0)"[1] 0508>id msg
program stack parameter msg: long @ [bp+8]

"go-calculator-v1(0)"[1] 0508>id pData
program stack parameter pData: ptr to ptr to struct CALC_ENGINE_DATA @ [bp+24]

"go-calculator-v1(0)"[1] 0508>id CalcEngineEnterOperator
program global CalcEngineEnterOperator:
function(
    msg: long, self: ptr to void, pArgs: ptr to struct CALC_ENGINE_TOKEN, ctx:
    ptr to void, pData: ptr to ptr to struct CALC_ENGINE_DATA, ...)
returning long @ go-calc_eng-v1(0) 1.926
**ids**

Display the identifiers known in a scope.

**Syntax**

```plaintext
ids [scopeSpec]
```

**scopeSpec** A scopeSpec as defined in Section 7.2.1. Defaults to the current scope and all of its parent scopes.

**Examples**

```
"go-calculator-v1(0)"[1] 0508>ids
{ (vars) pInst result s}
{ (params) msg self pArgs ctx pData}
{}
{ (tags) CALC_ENGINE_DATA CALC_ENGINE_NEW CALC_ENGINE_TOKEN OBJ_RESTORE
OBJ_SAVE OBJECT_NEW (vars) clsCalcEngineTable (functions) _8087
CalcEngineAppendCharToToken CalcEngineDump CalcEngineEnterNumber
CalcEngineEnterOperator CalcEngineEvalBinary CalcEngineEvalUnary
CalcEngineFree CalcEngineInit CalcEngineProcessKey CalcEngineRestore
CalcEngineSave CalcEngineSaveValue CalcEngineUpdateToken DllMain}
```

**Remarks**

The volume of information displayed by the ids command can be overwhelming. You might want to use the fns or vars commands to get a subset of the information displayed by ids.

---

**k**

Display the call stack.

The k command is a synonym for the st command; see the st command's Datasheet.

---

**log**

Starts and stops logging PenPoint debugging input and output.

**Syntax**

```plaintext
log [ [ + ] filename ]
```

**filename** The full pathname for the log file.

+ Optionally specifies that the log be appended to an existing file.

With no parameters, log turns off logging.

---

**mi**

Display memory information.

**Syntax**

```plaintext
mi
```

**Examples**

```
"go-calculator-v1(0)"[1] 0508>mi
allocated: 217088 task local 233472 task total 6328320 sys total
regions: 4 task local 5 task total 939 sys total (0 shr)
28672 task resident 184320 swapped
8257536 memory 7925760 allocatable 1372160 nonswappable 1597440 free
5242880 swap 2834432 free (page size = 4096)
```
mini
Enters PenPoint's mini-debugger.

Syntax
mini

Remarks
To return to DB, type g to the mini-debugger.

od
Send msgDump to an object.

Syntax
od object [pArgs]

object Expression that evaluates to an object.
pArgs pArgs sent with the message. Defaults to -1.

Examples
>od theRootWindow

msgDump(clsObject): object=theRootWindow {cls=[dyn 01c80094]}, caps=0x8618
parent=window has no parent, first child=[dyn 16f6034a](34)
bounds (x,y,w,h)=0,0,640,480
updateNesting=0, synchVer=6, tag=0
flags.style:
wsClipChildren wsClipSiblings
wsVisible wspaintable
Dirty region is:
411a12d8: yRange = 1 1, 1 x-range slots max, 0 slots allocated

>od theProcess

msgDump(clsObject): object=theProcess {cls=clsProcess}, caps=0x8018
msgDump(clsProcess): local well-known theProcess, process=0x0128

on
Creates a "handler" for specified events.

Most variations of the on command are needed only by advanced or specialized DB users. See the Advanced DB Techniques chapter for more information on the on command.

on access, on store
Causes execution to halt when memory is read and/or written.

Syntax
[taskSet] on store lvalue [eventAction]

taskSet A task set as defined in Section 7.2.7. Defaults to all tasks.
lvalue Typically a variable name, although it can also be any expression referencing a region of storage. (Consult any high quality C language reference manual for a more complete description.)
eventAction C code to be executed when the breakpoint is hit.

Examples
"go-calculator-v1(0)"[1] 0508>on store (*pData)->numberEntered
"go-calculator-v1(0)"[1] 0508>on access (*pData)->numberEntered
"go-calculator-v1(0)"[1] 0508>on access *(short *)043477c

Remarks
Only a limited amount of memory may be watched at any given time.
Execution halts after the instruction that reads and/or writes memory is executed. This is in contrast to breakpoints (created with the bp command), which halt execution before the breakpointed instruction.

There are other variations of the on command. See the Datasheet and the appropriate sections in the Advanced DB Techniques chapter.

**p, P**

Single-steps execution, passing over called routines.

p single-steps through source statements. P single-steps through assembly instructions.

**Syntax**

```plaintext
[taskSet] p [count]
[taskSet] P [count]
```

- **taskSet**: A task set as defined in Section 7.2.7. Defaults to the current task.
- **count**: Number of trace steps.

**Examples**

Executes one source statement.

```
"go-calculator-v1(0)":[1] 0508>p
78>> switch (op) {
```

Executes 10 source statements.

```
"go-calculator-v1(0)":[1] 0508>p 10
80>> if (opnd2 == 0.0) {
83>> *pResult = opnd1 / opnd2;
84>> }
104>> return s;
106>> } /* CalcEngineEvalBinary */
```

Executes one assembly instruction.

```
"go-calculator-v1(0)":[1] 0508>P
43A1C983 89 EC MOV ESP, EBP
```

Executes 10 assembly instructions.

```
"go-calculator-v1(0)":[1] 0508>P 10
43A1C985 5D POP EBP
43A1C986 C3 RET
E002D7FF 83 C4 14 ADD ESP, 20
E002D802 8D 65 F4 LEA ESP, [EBP-12]
E002D805 5F POP EDI
E002D806 5E POP ESI
E002D807 5B POP EBX
E002D808 5D POP EBP
E002D809 C2 0C 00 RET 12
43A1CABB 89 45 F4 MOV DWORD PTR [EBP-12(s)], EAX
```

**Remarks**

p and P differ from t and T in that p/P do not step into called routines while t/T step into routines.

Other un-frozen tasks may execute before the next source statement is reached.

If the p command is given when DB does not have source code information, it behaves just like the P command.
### profile
Create a profile breakpoint

**Remarks**
See Chapter 8, which discusses profiling and the **profile** command in detail.

```plaintext
q
```
Quit DB.

**Syntax**
```
q
```

```plaintext
r
```
Display the registers.

**Syntax**
```
[taskID] r
```
- **taskID** A task ID as defined in Section 7.2.7. Defaults to the current task.

**Examples**
```
"go-calculator-v1(0)"[1] 0508>r
eax 00000000 ebx 008010BC ecx 41D00031 edx 00430005 esi 43FE9538 edi 418F34F8
esp 00432C20 ebp 00432C2C eip 43A1CABB (user) ---0--IT---P--
```

### srcdir
Tells DB where to find source files for a module.

**Syntax**
```
srcdir [scopeSpec [dirName]]
```
- **scopeSpec** The identifier by which the module is known to PenPoint at load time; described in Section 7.2.1. Generally, you specify a .EXE, .DLL, .C, or .OBJ file.
- **dirName** The path to the directory containing the source for the module.

**Examples**
Specifies the source directory for the "calc_eng" module.
```
>srcdir "calc_eng" \c\penpoint\sdk\sample\calc
```
Without a dirName, srcdir gives the current source directory, if any, for a module.
```
>srcdir "calc_eng"
"calc.dll" \c\penpoint\sdk\sample\calc
```

**Remarks**
If you do not specify any arguments, srcdir shows all source directory settings.
Specifying "*" for the scopeSpec sets the default source directory for all modules not otherwise set with a srcdir command.

If DB finds no other srcdir set for a particular .C file, it checks the parent .OBJ file. If DB finds no srcdir set for a particular .OBJ file, it checks its parent .EXE file. If DB finds no srcdir set for a particular .EXE file, it uses the default srcdir.

Alternatively, you can set the DBSrc environment variable in \PENPOINT\BOOT\ENVIRON.INI. DB will search these directories after searching any directories specified in srcdir commands.
```
DBSrc=\c\dir1;\c\dir2;\c\dir3
```
Display the call stack.

**Syntax**

```st [taskID] st { [+ | ] [c | l | p] } [scopeSpec] [count]
```

- **taskID**: A task set as defined in Section 7.2.7. Defaults to the current task.
- **+/- c**: Controls display of context lines. These are the identifiers that are listed with varying indent to show their nesting levels.
- **+/- l**: Controls display of block-local variables.
- **+/- p**: Controls display of function parameters.
- **scopeSpec**: A scopeSpec as defined in Section 7.2.1.
- **count**: Number of stack frames to display.

The default flags are `+p -l -c`.

**Examples**

```
"go-calculator-v1"[1] 0508> st
%1 > CalcEngineEvalBinary(opnd1: 1.200000, op: 5, opnd2: 9.000000, pResult: 0432bbc)
%2 CalcEngineEnterOperator(
  msg: 25170108, self: 01ad103fb, pArgs: 043444c, ctx: 0432be8, pData: 043fe9538)
%3 ObjectCall [ msgCalcEngineEnterOperator [dyn 1ad103fb] 043444c 0 ]
%4 CalcEngineProcessKey(
  msg: 8392892, self: 01ad103fb, pArgs: 043444c, ctx: 0432c48, pData: 043fe9538)
%5 ObjectCall [ msgCalcEngineProcessKey [dyn 1ad103fb] 043444c 043a478d5 ]
%6 CalcAppButtonNotify(
  msg: 117440616, self: 01ac403e5, pArgs: 05, ctx: 0432ca0, pData: 043fe8e96)
%7 ObjectCall [ msgButtonNotify [dyn 1ac403e5] 05 01ad90402 043fe9b1c 0432d98 0409bc610 ]
```

```
"go-calculator-v1"[1] 0508> st + c p 1
%1 > (block) {s: 4409164i}
%2 (block) {pInst: 043474c; result: 2.132846e-307; s: 1140757816;}
CalcEngineEnterOperator(
  msg: 25170108, self: 01ad103fb, pArgs: 043444c, ctx: 0432be8, pData: 043fe9538)
%3 ObjectCall [ msgCalcEngineEnterOperator [dyn 1ad103fb] 043444c 0 ]
%4 (block) {s: 0;}
CalcEngineProcessKey{
```

```
..."go-calculator-v1"[1] 0508> st +1
%1 > (block) {s: 4409164i}
%2 (block) {pInst: 043474c; result: 2.132846e-307; s: 1140757816;}
CalcEngineEnterOperator(
  msg: 25170108, self: 01ad103fb, pArgs: 043444c, ctx: 0432be8, pData: 043fe9538)
%3 ObjectCall [ msgCalcEngineEnterOperator [dyn 1ad103fb] 043444c 0 ]
%4 (block) {s: 0;}
CalcEngineProcessKey{
```

```
..."go-calculator-v1"[1] 0508> st +1
%1 > (block) {s: 4409164i}
%2 (block) {pInst: 043474c; result: 2.132846e-307; s: 1140757816;}
CalcEngineEnterOperator(
  msg: 25170108, self: 01ad103fb, pArgs: 043444c, ctx: 0432be8, pData: 043fe9538)
%3 ObjectCall [ msgCalcEngineEnterOperator [dyn 1ad103fb] 043444c 0 ]
%4 (block) {s: 0;}
CalcEngineProcessKey{
```
With no parameters, \texttt{st} continues from where the previous \texttt{st} command left off.

The \texttt{st} command lists the execution contexts pushed onto the call stack for the current task up to the last instruction executed.

DB records function calls and moves into nested blocks on the call stack. Within each context, the variables local to it and any parameters defined at its entry are listed with their current values.

---

\textbf{sym}

Loads symbolic debugging information into memory.

\textbf{Syntax}

\texttt{sym \{ scopeSpec \[ (objFileList) \] \}}

\textit{scopeSpec} A scope spec that identifies a load module. Described in section 7.2.1.

\textit{objFileList} This is an optional list of comma-separated object file names (without the trailing ".obj"). If present, DB reads the detailed symbol information for only those object files, thereby saving substantial amounts of memory. The \texttt{objFileList} must be separated from the \texttt{scopeSpec} by a space. If \texttt{objFileList} consists of just "()" (the parentheses with no files between them), the command loads only the public identifiers and line numbers of the module.

\textbf{Examples}

The following three commands load all of the symbols in the module:

\texttt{>sym "foo"}

\texttt{>sym "go-foo-v1"}

The following command will load full symbolic debugging information for \texttt{obj1.obj} and \texttt{obj2.obj}, but only line numbers and global identifiers for the rest of the module's object files. (The space before the "(" is mandatory.)

\texttt{>sym "go-foo-v1" (obj1, obj2)}

\textbf{Remarks}

With no arguments, \texttt{sym} displays all programs and their associated symbol file, if any.
t, T

Single-steps execution, stepping into called routines.

\textit{t} single-steps through \textit{source} statements. \textit{T} single-steps through \textit{assembly} instructions.

**Syntax**

\[
\begin{align*}
\text{[taskSet]} & \quad \text{t} \quad \text{[count]} \\
\text{[taskSet]} & \quad \text{T} \quad \text{[count]}
\end{align*}
\]

- \textit{taskSet} A task set as defined in Section 7.2.7. Defaults to the current task.
- \textit{count} Number of trace steps.

**Examples**

Executes one source statement.

"go-calculator-v1(0)"[1] 0508>t
78>> switch (op) {

Executes 10 source statements.

"go-calculator-v1(0)"[1] 0508>t 10
80>> if (opnd2 == 0.0) {
83>> *pResult = opnd1 / opnd2;
84>> }
104>> return s;
106>> } /* CalcEngineEvalBinary */
418>> pInst->pendingOp = nop;
419>> CalcEngineSaveValue(result, pArgs, pInst);
161>> CalcEngineSaveValue (result, pInst);
166>> if (pInst->pendingOp == nop) {
167>> pInst->xValue = value;

Executes one assembly instruction.

"go-calculator-v1(0)"[1] 0508>T
43A1C3A7 8B 55 14 MOV EDX, DWORD PTR [EBP+20{pInst}]

Executes 10 assembly instructions.

"go-calculator-v1(0)"[1] 0508>T 10
43A1C3AA 89 02 MOV DWORD PTR [EDX], EAX
43A1C3AC 8B 45 0C MOV EAX, DWORD PTR [EBP+12(value+4)]
43A1C3AF 8B 55 14 MOV EDX, DWORD PTR [EBP+20(pInst)]
43A1C3B2 89 42 04 MOV DWORD PTR [EDX+4], EAX
43A1C3B5 8B 45 08 MOV EAX, DWORD PTR [EBP+8(value)]
43A1C3B8 8B 55 14 MOV EDX, DWORD PTR [EBP+20(pInst)]
43A1C3BB 89 42 08 MOV DWORD PTR [EDX+8], EAX
43A1C3BE 8B 45 0C MOV EAX, DWORD PTR [EBP+12(value+4)]
43A1C3C1 8B 55 14 MOV EDX, DWORD PTR [EBP+20(pInst)]
43A1C3C4 89 42 0C MOV DWORD PTR [EDX+12], EAX

**Remarks**

\textit{t} and \textit{T} differ from \textit{p} and \textit{P} in that \textit{t//T} step \textit{into} routines while \textit{p//P} do not step into called routines.

Other un-frozen tasks may execute before the next source statement is reached.

If the \textit{t} command is given when DB does not have source code information, it behaves just like the \textit{T} command.
**ti**

Display task information.

**Syntax**

\[ \text{[taskSet]} \text{ ti} \]

*taskSet* Set of tasks to show information for as described in Section 7.2.7. Defaults to the current task.

**Examples**

```
"go-calculator-v1(O)"[1] 0508>ti
00508 P "go-calculator-v1(O)"[1] 1.439 Step
```

**Remarks**

The ti command is just like the tl command, except that the default is different. See the Datasheet of the tl command for more information about ti's output.

---

**tl**

Display PenPoint's task list.

**Syntax**

\[ \text{[taskSet]} \text{ tl} \]

*taskSet* An optional task set as described in Section 7.2.7. DB displays the status of each task in the set. Defaults to all tasks.

**Examples**

```
>tl
00508 P "go-calculator-v1(O)"[1] 1.439 Step
004a8 P "go-nbtoc-v1"[1] .907 Msg
004d8 P "go-nbapp-v1"[1] .886 Msg
004c8 P "go-dtapp-v1"[1] 1.134 Msg
004b8 P "go-calculator-v1(O)"[0] .154 Msg
004a8 P "go-calc_eng-v1(O)"[0] .007 Susp Msg
...
00288 P "go-minitext-v1"[0] .372 Msg
00268 P "go-notepaperapp-v1"[0] .199 Susp Msg
00258 P "go-notepaper-v1(O)"[0] .041 Susp Msg
00238 P "go-nbapp-v1"[0] .126 Susp Msg
00218 P "go-nbtoc-v1(O)"[0] .127 Susp Msg
001f8 P "go-sectapp-v1"[0] .170 Susp Msg
```

**Remarks**

The first column of output displays PenPoint's task ID. The second column contains either a P for Process or S for Subtask. The third column displays DB's name for the task. The fourth column contains the accumulated runtime of the task. The fifth column contains a mnemonic indicating the state of the task. Common mnemonics and their meanings are:

- **Msg** Task is waiting for a message.
- **Sem** Task is waiting for a semaphore.
- **SSm** Task is waiting for a system semaphore.
- **FSm** Task is waiting for a fast semaphore.
- **Tmr** Task is waiting for a timer.
- **BP** Task is waiting for a breakpoint.
- **Step** Task is executing a single step.
- **Rdy** Task is ready to run.

The fifth column may also include the **Susp** keyword; if so, the task is suspended.
**type**
Display type that a C expression evaluates to.

**Syntax**

**C-expression**
Any C expression using variables that are defined in the current context, or any identifier known to the current context or fully referenced.

**Examples**

```c
>type 12
long
>type "hello"
array [6] of char
"go-calculator-v1(0)"[1] 0508>type self
ptr to void
"go-calculator-v1(0)"[1] 0508>type pData
ptr to ptr to struct CALC_ENGINE_DATA
"go-calculator-v1(0)"[1] 0508>type *pData
ptr to struct CALC_ENGINE_DATA
"go-calculator-v1(0)"[1] 0508>type **pData
struct CALC_ENGINE_DATA {
    xValue: double; yValue: double; pendingOp: short; keysSeen:
    array [30] of unsigned char; numberEntered: short; calcError: short;}
>type CalcEngineEvalBinary
function(opnd1: double, op: short, opnd2: double, pResult: ptr to double, ...)
returning long
```

**u**
View assembly code.

**Syntax**

```c
u [codeAddress | scopeSpec] [lineCount]
```

*codeAddress* A code address as defined in Section 7.2.3. Defaults to where the previous u command left off.

*scopeSpec* A scopeSpec as defined in Section 7.2.1.

*lineCount* A lineCount as defined in Section 7.2.2.

**Examples**

The following shows the assembly code that makes up the first part of the routine CalcEngineEvalBinary.

```
"go-calculator-v1(0)"[1] 0508>ctx top
"go-calculator-v1(0)"[1] 0508>u
43A1C19A C7 45 F8 00 00 00 00
43A1C1A1 66 8B 45 10 MOV AX, WORD PTR [EBP+16{op}]
43A1C1A5 66 83 E8 02 SUB AX, 2
43A1C1A9 66 3D 03 00 CMP AX, 3
43A1C1AD 0F 87 97 00 00 00 JA 043A1C24A{CalcEngineEvalBinary.8100}
43A1C1B3 0F B7 C0 MOVZX EAX, AX
43A1C1B6 C1 E0 02 SHL EAX, 2
43A1C1B9 2E FF A0 70 C1 A1 43 JMP DWORD PTR CS:[EAX-16016]
43A1C1C0 83 7D 18 00 CMP DWORD PTR [EBP+24{opnd2+4}], 0
43A1C1C4 75 0C JNZ 043A1C1D2{CalcEngineEvalBinary.803}
```

You can also specify a codeAddress for the u command.
"go-calculator-v1(0)"[1] 0508>u CalcEngineEvalBinary 1 10
43A1C195 55 PUSH EBP
43A1C196 89 E5 MOV EBP, ESP
43A1C198 53 PUSH EBX
43A1C199 50 PUSH EAX
43A1C19A C7 45 F8 00 00 00 00 MOV DWORD PTR [EBP-8{s}], 0
43A1C1A1 66 8B 45 10 MOV AX, WORD PTR [EBP+16{op}]
43A1C1A5 66 83 E8 02 SUB AX, 2
43A1C1A9 66 3D 03 00 CMP AX, 3
43A1C1AD 0F 87 97 00 00 00 00 JA 043A1C24A{CalcEngineEvalBinary.@100}
43A1C1B3 0F B7 C0 MOVZX EAX, AX

You should not specify a data address; you will get spurious results
Where possible, the u command inserts an identifier after addresses that it can interpret symbolically.
Repeating the u command displays the next several lines of source code. In this way you can page down through a source file.

uv
View assembly code with interspersed source code.

Syntax
The syntax of the uv command is identical to that of the u command.

Examples
The following shows the assembly code that makes up the first part of the routine CalcEngineEvalBinary, with source code statements interspersed.

"go-calculator-v1(0)"[1] 0508>ctx top
"go-calculator-v1(0)"[1] 0508>uv
    76     STATUS          s = stsOK;
    77
43A1C19A C7 45 F8 00 00 00 00 MOV DWORD PTR [EBP-8{s}], 0
    78     switch (op) {
    79         case divide:
43A1C1A1 66 88 45 10 MOV AX, WORD PTR [EBP+16{op}]
43A1C1A5 66 83 E8 02 SUB AX, 2
43A1C1A9 66 3D 03 00 CMP AX, 3
43A1C1AD 0F 87 97 00 00 00 00 JA 043A1C24A{CalcEngineEvalBinary.@100}
43A1C1B3 0F B7 C0 MOVZX EAX, AX
43A1C1B6 C1 E0 02 SHL EAX, 2
43A1C1B9 2E FF A0 70 C1 A1 43 JMP DWORD PTR CS:[EAX-16016]
    80         if (opnd2 == 0.0) {
43A1C1C0 83 7D 18 00 CMP DWORD PTR [EBP+24{opnd2+4}], 0
43A1C1C4 75 0C JNZ 043A1C1D2{CalcEngineEvalBinary.@83}

You can also specify a codeAddress for the uv command.

"go-calculator-v1(0)"[1] 0508>uv CalcEngineEvalBinary 1 10
    70     CalcEngineEvalBinary ( 
    71         double          opnd1,
72         CALC_ENGINE_KEY op,
73         double          opnd2,
74         double *        pResult)
    75     )
43A1C195 55 PUSH EBP
43A1C196 89 E5 MOV EBP, ESP
43A1C198 53 PUSH EBX
43A1C199 50 PUSH EAX
    76     STATUS          s = stsOK;
The mapping between assembly code and source code is, by its nature, imprecise. It is particularly imprecise when the compiler's optimization is turned on.

The uv command differs from the vu command in that the uv command always list the assembly code properly and intersperses the source code as best it can given the imprecise nature of the task.

See Remarks in the u Datasheet.

**v**

View source code.

**Syntax**

\textbf{v} \text{ [codeAddress} \space | \text{ scopeSpec} \space | \text{ lineCount} ]

- **codeAddress**: A code address as defined in Section 7.2.3. Defaults to where the previous v command left off.
- **scopeSpec**: A scopeSpec as defined in Section 7.2.1.
- **lineCount**: A lineCount as defined in Section 7.2.2.

**Examples**

"go-calculator-v1(0)" [1] 0508>ctx top
"go-calculator-v1(0)" [1] 0508>v

73 \hspace{1em} \text{double opnd2,}
74 \hspace{1em} \text{double * pResult} )
75 \{ 
76 \hspace{1em} \text{STATUS} \quad s = \text{stsOK};
77 
78 \hspace{1em} \text{switch (op) } \{ 
79 \hspace{1em} \text{case divide:}
80 \hspace{1em} \text{if (opnd2 == 0.0) } \{
81 \hspace{1em} \text{s = \text{stsCalcEngineComputeError};}
82 \hspace{1em} \text{else } \{ 
83 \hspace{1em} \text{*pResult = opnd1 / opnd2;}
84 \hspace{1em} \}
85 \hspace{1em} \text{break;}
86 
"go-calculator-v1(0)" [1] 0508>ctx top
"go-calculator-v1(0)" [1] 0508>v l 10

76 \hspace{1em} \text{STATUS} \quad s = \text{stsOK};
77 
78 \hspace{1em} \text{switch (op) } \{ 
79 \hspace{1em} \text{case divide:}
80 \hspace{1em} \text{if (opnd2 == 0.0) } \{
81 \hspace{1em} \text{s = \text{stsCalcEngineComputeError};}
82 \hspace{1em} \text{else } \{
83 \hspace{1em} \text{*pResult = opnd1 / opnd2;}
84 \hspace{1em} \}
85 \hspace{1em} \text{break;}
86

**Remarks**

Repeating the v command displays the next several lines of source code. In this way you can page down through a source file.
**vars**

Display variables known in a scope.

**Syntax**

vars [scopeSpec]

*scopeSpec* A *scopeSpec* as defined in Section 7.2.1. Defaults to showing the variables in the current scope and each of its parent scopes. If you specify "top", vars displays the variables declared interactively with the debugger.

**Examples**

```
"go-calculator-v1(0)"[1] 0508>vars
{(vars) s}
{(params) opndl op opnd2 pResult}
{(vars) clsCalcEngineTable}

"go-calculator-v1(0)"[1] 0508>vars $6
{(vars) s}
{(params) msg self pArgs ctx pData}
```

**Remarks**

This command displays a subset of the information displayed by the ids command.

---

**ver**

Display DB's version.

**Syntax**

ver

**vu**

View source code with interspersed assembly language statements.

**Syntax**

The syntax of the vu command is identical to that of the v command.

**Examples**

```
"go-calculator-v1(0)"[1] 0508>ctx top
"go-calculator-v1(0)"[1] 0508>vu 1 4

76>> STATUS s = ststOK;
43A1C19A C7 45 F8 00 00 00 00
    MOV DWORD PTR [EBP-B{s}], 0

77
43A1C19A C7 45 F8 00 00 00 00
    MOV DWORD PTR [EBP-B{s}], 0

78    switch (op) {
43A1C1A1 66 8B 45 10    MOV AX, WORD PTR [EBP+16{op}]
43A1C1A5 66 83 E8 02    SUB AX, 2
43A1C1A9 66 3D 03 00    CMP AX, 3
43A1C1AD 0F 87 97 00 00 00    JA 043A1C24A{CalcEngineEvalBinary.@100}
43A1C1B3 0F B7 C0    MOVZX EAX, AX
43A1C1B6 C1 E0 02    SHL EAX, 2
43A1C1B9 2E FF A0 70 C1 A1 43    JMP DWORD PTR CS:[EAX-16016]

79    case divide:
43A1C1A1 66 8B 45 10    MOV AX, WORD PTR [EBP+16{op}]
43A1C1A5 66 83 E8 02    SUB AX, 2
43A1C1A9 66 3D 03 00    CMP AX, 3
43A1C1AD 0F 87 97 00 00 00    JA 043A1C24A{CalcEngineEvalBinary.@100}
43A1C1B3 0F B7 C0    MOVZX EAX, AX
43A1C1B6 C1 E0 02    SHL EAX, 2
43A1C1B9 2E FF A0 70 C1 A1 43    JMP DWORD PTR CS:[EAX-16016]
```
The mapping between assembly code and source code is, by its nature, imprecise. It is particularly imprecise when the compiler's optimization is turned on.

The `vu` command differs from the `uv` command in that the `vu` command always lists the source code properly and intersperses the assembly code as best it can given the imprecise nature of the task.

See Remarks in the `v` Datasheet.

---

**zp**

Clears profile data.

Remarks

See Chapter 8, which discusses profiling and the `zp` command in detail.
Chapter 8 / Profiling with DB

This chapter contains information about using DB to profile program execution. This can help you find performance bottlenecks in your program.

Profile Breakpoints

Profile information is maintained in profile breakpoints. Many breakpoint commands operate on profile breakpoints as well as normal breakpoints; consult the appropriate Datasheet for complete information.

- bl
- bc
- bd
- be

Two Types of Profiles

DB can collect two types of profiles:

- Code Profiles This type of profile collects execution information based on routines or line numbers.
- Object Profile This type of profile collects execution information based on objects and messages.

With both types, DB collects information into profiling buckets. You can control the number (or granularity) of the buckets and what gets recorded in each bucket.

Code Profiling

Code profiling collects information about routines or lines of code.

There are two code profiling techniques; each has advantages and disadvantages:

- Timing/Counting. This technique measures the time spent in a routine or counts the number of times a routine is executed.
  - Advantage: Gives very accurate information.
  - Disadvantage: Can significantly slow program execution.

- Sampling. This technique samples the CPU’s instruction pointer periodically and records which routine the pointer is in.
  - Advantage: Far less intrusive; the performance overhead is imperceptible.
  - Disadvantage: Sampled data, by its nature, is less accurate.
  - Disadvantage: The technique can tell you where time is being spent, but not why. For instance, if your program spends most of its time in
the file system, the samples themselves will all be in the file system. You won't know which of your routines is responsible for the file system calls.

As illustrated by the examples later in this chapter, you'll probably want to use each of these techniques under different circumstances.

**Code Profiling Options**

The syntax for specifying a code profile is:

```
[taskSet] profile [flags] routineSet
```

The `taskSet` defaults to all tasks.

The `flags` determine what gets recorded in each bucket and what profiling technique is used. The flags are one of the following:

- `$t$` Timing. The breakpoint measures how much time is spent in the routine.
- `$c$` Counting. The breakpoint counts the number of times the routine is entered. This is the default.
- `$h$` Sampling. The breakpoint records how often the CPU's instruction pointer is in the routine.

Both `$t$` and `$c$` can be on in a single profile.

The default flags are `$+c -t -h$`.

The `routineSet` determines the buckets that are created for a code profile; DB collects information for each bucket. A routine set is a set of routines or line numbers to be profiled. There are several ways to specify a routine set, as described in the following table.

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>scopeSpec</td>
<td>A scope specification for an installed .EXE, .DLL or .OBJ file. The scope specification may be followed by one of the following:</td>
</tr>
<tr>
<td>publics</td>
<td>One bucket for each public routine.</td>
</tr>
<tr>
<td>statics</td>
<td>One bucket for each static routine.</td>
</tr>
<tr>
<td>all</td>
<td>One bucket for each routine. (This is the default.)</td>
</tr>
<tr>
<td>lines</td>
<td>One bucket for each executable line.</td>
</tr>
<tr>
<td>exes</td>
<td>One bucket for each installed .EXE and .DLL.</td>
</tr>
<tr>
<td>routineIdentifier</td>
<td>One bucket created for the routine identified with a Scope.Identifier reference, as discussed in Section 7.2.6.</td>
</tr>
<tr>
<td>lines</td>
<td>If followed by the optional :lines, then one bucket is created for each line in the routine.</td>
</tr>
<tr>
<td>routineSet +/- routineSet</td>
<td>Union or difference of two routine sets.</td>
</tr>
</tbody>
</table>

1. Technically, one bucket is created for each code segment of each installed .EXE and .DLL. But it is very rare for an .EXE or .DLL to have more than one code segment.
A Caveat Concerning Sampling Profiles

If DB does not have full symbols for a module being code-profiled, it only knows the start of each public function. Thus if you use

```
profile +h "foo.exe"
```

and your source looks like this:

```c
STATUS A(void) {...}
static STATUS B(void) {...}
STATUS C(void) {...}
static STATUS D(void) {...}
STATUS E(void) {...}
```

The samples found in B and D will be charged to A and C. If you use

```
profile +c "foo.exe"
```

B and D will be ignored altogether.

You can use these techniques to reduce the impact of this problem:

- When using sample-based code profiles be sure that the modules you profile were compiled with the /D2 flag and were linked with the DEBUG ALL line. Code compiled and linked in this fashion provides DB with the start addresses of all the routines, public or private, so DB is much more likely to attribute a sample to the proper routine.

- Use the STATIC and LOCAL keywords (defined in GO.H) rather than the C language construct static, and compile your code with the DEBUG pre-processor variable defined. Under these conditions, the STATIC and LOCAL routines will actually be public routines rather than private routines. You can then compile with /D1 and link with DEBUG ALL. Now DB will have the start location of every public and pseudo-static routine.

Getting More Frequent Samples

You can control the sampling rate. Decreasing the times between samples will increase accuracy but will increase the profiling overhead. The default rate is one sample per 55 milliseconds. You can set that to, say, one sample per 11 milliseconds (probably the smallest reasonable value) by typing:

```
?_SetSystick(11)
```

Code Profiling Examples

Sampling Profiles: A First Step

To get a rough idea of what some operation is spending its time in, create a profiling breakpoint as follows:

```
> profile +h exes
> g
```

Now perform the operation one or more times. Remember that the sampling rate is a bit coarse (approximately 55 milliseconds), so you want a reasonable number
of samples. Finally, get back to DB by typing `PAUSE` and type the following, which lists the time spent in each module, as estimated from the samples taken:

```
> dp 1
```

**Refining the Profile with Smaller Buckets**

Suppose you’ve found that all your time is spent in the "foo" module. You can set up a sampling profile with one bucket per function by saying:

```
> sym "foo"
> profile +h "foo"
```

Or, for just the functions in FILE1.OBJ:

```
> profile +h "foo"."file1.obj"
```

Or, for every source line in FILE1.OBJ:

```
> profile +h "foo"."file1.obj":lines
```

Or, for every source line in the function `SomeFunction`:

```
> profile +h "foo".SomeFunction:lines
```

**Refining the Profile with Infinite Buckets**

If you use the following profile, DB will record every sample, and the `dp` command will then group the samples by function (where it knows symbols) and by .EXE and .DLL (where it doesn’t).

```
> profile +h *
```

**Timing/Counting Profiles**

With the following profile, DB will count the number of times each public routine in module "foo" is executed. (Notice the `+c` rather than the `+h` used in the previous examples.)

```
> profile +c "foo"
```

With the following profile, DB will accumulate the time spent in each public routine in module "foo". (Notice the `+t` rather than the `+h` and `+c` used in the previous examples.)

```
> profile +t "foo"
```

DB can accumulate both time and count information: The `dp` command will then show you how much time was spent per call.

```
> profile +t +c "foo"
```

With the following, DB will accumulate the time spent in each line in module "foo."

```
> profile +t "foo":lines
```
If specific routines seem to be using the most time, you can add line-by-line hit counts for them by specifying:

```plaintext
> profile "foo":all + "foo".RoutineA:lines + "foo".RoutineB:lines
```

### Object Profiling

Object profiling collects information about messages sent via `ObjectCall`.

#### Basic Object Profiling

The default object profile records and counts all messages sent to all objects:

```plaintext
> profile oc *
```

### Object Profiling Options

The general syntax for creating an object profile is:

```plaintext
[taskSet] profile [flags] oc messagePattern
```

Notice the `-oc`; this flag tells DB to create an Object profile rather than a Code profile.

The following flags control the number of buckets:

- `+/- msg` One bucket per message encountered. (This is the default.)
- `+/- msgCls` One bucket per class of message encountered.
- `+/- obj` One bucket per receiver object encountered.
- `+/- objCls` One bucket per class of message receiver encountered.

The following flags control what gets recorded in each bucket:

- `+/- c` Counts the number of messages sent. (This is the default.)
- `+/- t` Records execution time.

The default flags are `+msg +c`.

### Object Profiling Message Pattern

The message pattern defines what messages are of interest; only messages which match the pattern are recorded. There are several ways to specify a message pattern, as described in the following tables.

#### Table 8-2: Message Pattern Specification

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>msgList</code></td>
<td>Any message in the list matches the pattern.</td>
</tr>
<tr>
<td>to <code>objectList</code></td>
<td>Any message sent to any object in the list matches the pattern.</td>
</tr>
<tr>
<td><code>msgList</code> to <code>objectList</code></td>
<td>Any message in the list sent to any object in the list matches the pattern.</td>
</tr>
<tr>
<td><code>messagePattern</code></td>
<td>Union or difference of two message patterns.</td>
</tr>
</tbody>
</table>
A `msgList` is specified as follows:

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>*</code></td>
<td>Any message.</td>
</tr>
<tr>
<td><code>msgFoo</code></td>
<td>The message with the string name <code>msgFoo</code>.</td>
</tr>
<tr>
<td><code>clsFoo</code></td>
<td>All messages defined by the class <code>clsFoo</code>.</td>
</tr>
<tr>
<td><code>C expression</code></td>
<td>The message that has the value of <code>C expression</code>.</td>
</tr>
<tr>
<td><code>C expression:msgs</code></td>
<td>All messages defined by the class that has the value of <code>C expression</code>.</td>
</tr>
<tr>
<td><code>msgList +/- msgList</code></td>
<td>Union or difference of two message lists.</td>
</tr>
<tr>
<td><code>[msgList]</code></td>
<td>A message list in required square brackets.</td>
</tr>
</tbody>
</table>

1. Technically, all messages `msgX` for which `ClsNum(msgX) == ClsNum(clsFoo)`.  
2. Technically, all messages `msgX` for which `ClsNum(msgX) == ClsNum(C expression)`.

An `objectList` is specified as follows:

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>*</code></td>
<td>Any object.</td>
</tr>
<tr>
<td><code>objectName</code></td>
<td>The object with the string name <code>objectName</code>. (Normally these will be well-known objects.)</td>
</tr>
<tr>
<td><code>clsFoo:objs</code></td>
<td>All objects of class <code>clsFoo</code> or a subclass of <code>clsFoo</code>.</td>
</tr>
<tr>
<td><code>C expression</code></td>
<td>The object whose uid has the value of <code>C expression</code>.</td>
</tr>
<tr>
<td><code>C expression:objs</code></td>
<td>All objects whose class (or some superclass) is the class that has the value of <code>C expression</code>.</td>
</tr>
<tr>
<td><code>objectList +/- objectList</code></td>
<td>Union or difference of two object lists.</td>
</tr>
<tr>
<td><code>[objectList]</code></td>
<td>An object list in required square brackets.</td>
</tr>
</tbody>
</table>

1. Technically, call objects `objX` for which `ObjectCall(msgIsA, objX, expression) == stsOK`.

### Object Profiling Examples

- The following command records all `ObjectCalls` to any object that `IsA clsWin`:
  ```
  > profile oc clsWin:objs
  ```
- The following records all `ObjectCalls` to `clsWin` itself:
  ```
  > profile oc clsWin
  ```
- The following records all messages in which the class of the message is `clsWin`:
  ```
  > profile oc clsWin:msgs
  ```
- The following records all messages in task 0508 in which the class of the message is `clsWin` or `clsButton`:
  ```
  > 0508 profile oc clsWin:msgs + clsButton:msgs
  ```
Displaying Profile Information: the DP Command

The dp command displays the current profile information for a profile breakpoint.

```
dp [flags] breakpoint
```

flags Described below.

breakpoint DB’s breakpoint number as displayed by the bl command.

The flags specify what columns get printed, and their order determines how the rows are sorted. flags are a sequence of the following:

<table>
<thead>
<tr>
<th>Entry</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>+/- a</td>
<td>code address (code profiles only)</td>
</tr>
<tr>
<td>+/- c</td>
<td>count</td>
</tr>
<tr>
<td>+/- cpct</td>
<td>displays the percent of the total accumulated hits</td>
</tr>
<tr>
<td>+/- fns</td>
<td>group all addresses by functions (useful with routine sets that include :lines or *, if symbolic debugging information is loaded)</td>
</tr>
<tr>
<td>+/- h</td>
<td>systick time (code profiles only)</td>
</tr>
<tr>
<td>+/- msg</td>
<td>message</td>
</tr>
<tr>
<td>+/- msgCls</td>
<td>message class</td>
</tr>
<tr>
<td>+/- obj</td>
<td>object</td>
</tr>
<tr>
<td>+/- objCls</td>
<td>receiver object class</td>
</tr>
<tr>
<td>+/- t</td>
<td>time</td>
</tr>
<tr>
<td>+/- tpc</td>
<td>time per count</td>
</tr>
<tr>
<td>+/- tpct</td>
<td>like +t but displays the percent of the total recorded times (time in function divided by time in all profiled functions)</td>
</tr>
<tr>
<td>+/- tpctt</td>
<td>like +tpct but displays the percent of the total runtime of the task(s) being profiled (time in function divided by time executing anywhere)</td>
</tr>
<tr>
<td>+/- z</td>
<td>include entries with a 0 value</td>
</tr>
</tbody>
</table>

The default flags display everything that was recorded and sort by time (or by time/count if that can be computed).

DP Command Examples

To display the information stored in profile breakpoint 1, type the following.

(This profile was collected by tapping three number buttons on the Calculator.)

```
>0508 profile oc clsWin:msgs + clsButton:msgs
>g
>dp 1

#hits msg
18 msgWinGetMetrics
12 msgWinBeginPaint
12 msgWinEndPaint
12 msgWinSend
9 msgWinGetFlags
9 msgWinUpdate
```
Here are several more examples of displaying the same profile information with several different flags.

```
>profile +c +t "calceng"
>g
>dp 1
#secs/hit  #seconds  #hits  addr
.007115   .099610  14  CalcEngineProcessKey.@443
.006520   .045646  7   CalcEngineEnterOperator.@377
.005877   .041145  7   CalcEngineAppendCharToToken.@200
.005833   .011666  2   CalcEngineEnterNumber.@359
.002379   .014274  6   CalcEngineSaveValue.@166
.002083   .006250  3   CalcEngineEvalUnary.@119
.001733   .015601  9   CalcEngineUpdateToken.@183
.001546   .001546  1   CalcEngineEvalBinary.@76
```

```
>dp +t 1
#seconds  addr
.099610  CalcEngineProcessKey.@443
.045646  CalcEngineEnterOperator.@377
.041145  CalcEngineAppendCharToToken.@200
.014274  CalcEngineSaveValue.@166
.011666  CalcEngineEnterNumber.@359
.006250  CalcEngineEvalUnary.@119
.001546  CalcEngineEvalBinary.@76
```

```
>dp +tpct 1
#hits  addr
42.253  CalcEngineProcessKey.@443
19.363  CalcEngineEnterOperator.@377
17.453  CalcEngineAppendCharToToken.@200
6.618   CalcEngineUpdateToken.@183
6.055   CalcEngineSaveValue.@166
4.948   CalcEngineEnterNumber.@359
2.651   CalcEngineEvalUnary.@119
.655    CalcEngineEvalBinary.@76
```

To display the time sorted by address:

```
>dp +a +t 1
```

To include zero times:

```
>dp +a +t +z 1
```
Clearing Profile Information: the ZP Command

The `zp` command clears profile data for a profile breakpoint. This allows you to zero out any accumulated profiling information and then start another run, collecting another profile.

```plaintext
zp breakpoint

breakpoint  DB's breakpoint number as displayed by the `bl` command.
```

Profiling Specific Tasks

By default, a profile records the execution of all tasks. You can limit profiling to a task by specifying the task ID:

```plaintext
>01228 profile ...
```

You can also limit profiling to all tasks executing a specific application:

```plaintext
>"someapp" profile ...
```
Chapter 9 / Advanced DB Techniques

This chapter discusses some advanced or specialized debugging techniques and the DB features that support them. Most of the material in this chapter should be needed rarely, if ever, by a typical PenPoint developer.

**Skipping Execution**

DB can skip over pieces of code. This is useful if you know that some piece of code is buggy, and you'd like to skip over it rather than execute it.

In this example, assume that the utility routine called in line 437 will cause a fatal problem:

```c
415    MsgHandler (TEAppCaseCmd)
416    {
417        STATUS s = stsOK;
418        U32 gType;
419        Dbg(Debugf("Case command self=0x%lx pArgs=0x%lx",self,pArgs));
420        switch ((TE_CASE_STYLE) (U32) pArgs) {
421            case teCaseStyleToUpper:
422                gType = xgsRightUpFlick;
423                break;
424            case teCaseStyleToLower:
425                gType = xgsRightDown;
426                break;
427            case teCaseStyleToInitialCaps:
428                gType = xgsRightUp;
429                break;
430            default:
431                s = stsBadParam;
432        }
433        if (s == stsOK) {
434            s = SomeUtilityRoutine(gType, self);
435        }
436        DbgTEAppCaseCmd("returns 0x%lx",(S32)s))
437        return s;
```

We can skip the execution of that routine with the following breakpoint:

```plaintext
> bp "minitext.exe"."teusr.obj".@420 {
    !?EIP="teusr.c".@439;
} }
```

Note that this breakpoint is set on line 420, not on the entry to the routine, so that the initialization of the status variable will occur.
The breakpoint code says, in effect, that when the breakpoint is hit, that the instruction point (EIP) should be set to line 439, and that execution should continue.

**Controlling Threads of Execution**

You can control multiple execution threads so that background tasks will not be able to change variables while you trace execution in a selected task. This is useful when debugging programs that have a significant amount of multi-processing in them.

There are three commands that control threads of execution:

**The FZ ("freeze") Command**

The fuzz command increments the freeze count on the specified task(s). A task will not execute while its freeze count is greater than 0.

```
[taskSet] fz
```

*taskSet* A task set as defined in Section 7.2.7. Defaults to the current task.

**The TH ("thaw") Command**

The th command thaws (decrements) the freeze count of the specified task(s). If the freeze count of the task becomes 0, the task is able to run.

```
[taskSet] th
```

*taskSet* A task set as defined in Section 7.2.7. Defaults to the current task.

**The TT ("terminate task") Command**

The tt command terminates a task or tasks.

```
[taskSet] tt
```

*taskSet* Set of tasks to be terminated. Defaults to the current task.

**Commands Executed at Compile Time**

When a piece of code is attached to a breakpoint (or any DB event handler), the code is compiled into an internal form for later execution.

Some DB commands execute as they are being entered into a piece of code. Therefore these commands are not available for execution inside of a breakpoint's attached code.

The DB commands that execute at compile time are:

```
<  log
h  mini
id qsrcdir
ids type
files vars
fns ver
```
DB Built-ins

For advanced use, DB has many useful built-in types, variables, and routines.

**DB’s Predefined Types**

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOLEAN, P_BOOLEAN</td>
<td>as in go.h</td>
</tr>
<tr>
<td>S8, P_S8</td>
<td>as in go.h</td>
</tr>
<tr>
<td>U8, P_U8</td>
<td>as in go.h</td>
</tr>
<tr>
<td>CHAR, P_CHAR</td>
<td>as in go.h</td>
</tr>
<tr>
<td>S16, P_S16</td>
<td>as in go.h</td>
</tr>
<tr>
<td>U16, P_U16</td>
<td>as in go.h</td>
</tr>
<tr>
<td>S32, P_S32</td>
<td>as in go.h</td>
</tr>
<tr>
<td>U32, P_U32</td>
<td>as in go.h</td>
</tr>
<tr>
<td>SIZEOF CONTEXT</td>
<td>as in go.h</td>
</tr>
<tr>
<td>P_ARGS</td>
<td>as in go.h</td>
</tr>
<tr>
<td>_HEX_LONG</td>
<td>a U32 that the ? command displays in hex</td>
</tr>
<tr>
<td>_HEX_SHORT</td>
<td>a U16 that the ? command displays in hex</td>
</tr>
<tr>
<td>_HEX_BYTE</td>
<td>a U8 that the ? command displays in hex</td>
</tr>
<tr>
<td>_BITS</td>
<td>an enum with values b0=1, b1=2, b2=4, etc</td>
</tr>
<tr>
<td>MESSAGE, MSG</td>
<td>a U32 that the ? command displays as a message name</td>
</tr>
<tr>
<td>OBJECT, OBJ</td>
<td>a U32 that the ? command displays as an object name</td>
</tr>
<tr>
<td>STATUS, STS</td>
<td>a U32 that the ? command displays as a status name</td>
</tr>
<tr>
<td>_MSG_PAT</td>
<td>a message pattern; see below.</td>
</tr>
<tr>
<td>_ROUTINE_SET</td>
<td>a routine set; see below.</td>
</tr>
<tr>
<td>_TASK_SET</td>
<td>a task set; see below.</td>
</tr>
</tbody>
</table>

The last three types can be used to declare variables that can be set and then used as parameters to certain DB commands. This is particularly useful because values of these types can potentially be very complex.

Here’s an example of a _MSG_PAT variable being used as a parameter to the profile command:

```c
{    _MSG_PAT mpl = [mp clsWin:msgs];    profile +c oc (mpl); }
```

Here’s an example of a _ROUTINE_SET variable being used as a parameter to the profile command:

```c
{    _ROUTINE_SET rsl = [rs "winc"+"win0":lines];    profile +c (rsl); }
```
Here's an example of a _TASK_SET variable being used as a parameter to the tl command:

```c

```{ TASK_SET tsl = [ts "minitext"+"browser"]; (tsl] tl; }

### Useful Values in DB

The values in the following table are occasionally useful. In general, you should not set the values.

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TASK_SET</td>
<td>_allITs</td>
<td>The task set containing all tasks.</td>
</tr>
<tr>
<td>TASK_SET</td>
<td>_allProgTS</td>
<td>The task set containing all applications (that is, all processes running .exes).</td>
</tr>
<tr>
<td>U32</td>
<td>_nTicks</td>
<td>The total elapsed systicks.</td>
</tr>
<tr>
<td>_RUNTIME</td>
<td>_runtime</td>
<td>The total run time over all tasks.</td>
</tr>
<tr>
<td>_T_REGS</td>
<td>regs</td>
<td>The current task's registers, as a structure.</td>
</tr>
<tr>
<td>_HEX_LONG</td>
<td>eip, eax, ebx,...</td>
<td>The current task's registers, individually.</td>
</tr>
<tr>
<td>_FLAGS</td>
<td>efg</td>
<td>The current task's flags.</td>
</tr>
<tr>
<td>_HEX_LONG</td>
<td>esp0</td>
<td>The current task's ring-0 stack pointer.</td>
</tr>
<tr>
<td>_HEX_SHORT</td>
<td>cs</td>
<td>The current task's code-segment register.</td>
</tr>
<tr>
<td>OS_TASK_ID</td>
<td>processId</td>
<td>The current task's process id.</td>
</tr>
<tr>
<td>U16</td>
<td>instanceNum</td>
<td>The current task's instance number.</td>
</tr>
<tr>
<td>OS_TASK_ID</td>
<td>_taskId</td>
<td>The victim task; irrelevant in an “on intReq” handler.</td>
</tr>
<tr>
<td>_TASK_FRAME</td>
<td>_taskFrame</td>
<td>The resume address and top frame pointer of &lt;_taskId&gt;.</td>
</tr>
<tr>
<td>_HANDLER_NUMBER</td>
<td>_handlerN</td>
<td>The handler number of the current handler.</td>
</tr>
<tr>
<td>_P_UNKNOWN</td>
<td>_pData</td>
<td>The handler data of the current handler.</td>
</tr>
<tr>
<td>_PARSED_PROG_NAME</td>
<td>_ppn</td>
<td>If “on installBegun”, “on installDone,” “on start,” the name of the program, parsed into its components: vendor-program-major(minor).</td>
</tr>
<tr>
<td>MESSAGE</td>
<td>msg</td>
<td>If “on oc,” the message.</td>
</tr>
<tr>
<td>OBJECT</td>
<td>self</td>
<td>If “on oc,” the message recipient.</td>
</tr>
<tr>
<td>P_ARGS</td>
<td>pArgs</td>
<td>If “on oc,” a pointer to the message arguments.</td>
</tr>
</tbody>
</table>
DB's Useful Variables

The variables in the next table can be examined and set.

Here's an example of setting the _keepTypedefs variable:

```>
!_keepTypedefs = true;
```

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOLEAN</td>
<td>_showRuntime</td>
<td>false</td>
<td>If true, g, t, p, etc., show the task runtime.</td>
</tr>
<tr>
<td>BOOLEAN</td>
<td>_showRealtime</td>
<td>false</td>
<td>If true, g, t, p, etc., show the elapsed time.</td>
</tr>
<tr>
<td>BOOLEAN</td>
<td>_suspCur</td>
<td>true</td>
<td>DB suspends the current task while in DB command mode; if false, DB lets it run.</td>
</tr>
<tr>
<td>BOOLEAN</td>
<td>_redirectInput</td>
<td>true</td>
<td>When not in command mode, DB redirects input to the PenPoint input system; if false, such input is just DB typeahead.</td>
</tr>
<tr>
<td>BOOLEAN</td>
<td>_redirectPrinting</td>
<td>false</td>
<td>If true, remote DB redirects Debug() output to the host.</td>
</tr>
<tr>
<td>BOOLEAN</td>
<td>_keepUnusedTypes</td>
<td>false</td>
<td>DB discards any types from an .exe/dll that are not actually &quot;reachable&quot; from some function or variable definition.</td>
</tr>
<tr>
<td>BOOLEAN</td>
<td>_keepTypedefs</td>
<td>false</td>
<td>If false, DB discards any typedef definitions from an .exe/.dll.</td>
</tr>
<tr>
<td>_TASK_SET</td>
<td>_starTS</td>
<td>_pAllProgTS</td>
<td>The task set you get when you say &quot;[*]&quot;, and the default task set for commands like bp, tl, etc.</td>
</tr>
<tr>
<td>_TASK_SET</td>
<td>_suspTS</td>
<td>empty set</td>
<td>DB suspends all these tasks (in addition to the current task) whenever it enters command mode.</td>
</tr>
<tr>
<td>S8</td>
<td>_defCodeMode</td>
<td>1 ('l')</td>
<td>Mode of u command. Currently the only legal value is 1.</td>
</tr>
<tr>
<td>U16</td>
<td>_defCodeLength</td>
<td>8</td>
<td>Number of lines output by u command.</td>
</tr>
<tr>
<td>S8</td>
<td>_defViewMode</td>
<td>0 ('a')</td>
<td>Mode of v command. May also be -1 ('b') or 1 ('l').</td>
</tr>
<tr>
<td>U16</td>
<td>_defViewLength</td>
<td>8</td>
<td>Number of lines output by v command.</td>
</tr>
<tr>
<td>S8</td>
<td>_defDataMode</td>
<td>1 ('l')</td>
<td>Mode of d, db, dw, and dd commands. May also be -1 ('b') or 0 ('a').</td>
</tr>
<tr>
<td>U16</td>
<td>_defDataLength</td>
<td>64</td>
<td>Length (in bytes) for d, db, dw, and dd commands.</td>
</tr>
<tr>
<td>S8</td>
<td>_defShowStackMode</td>
<td>0 ('a')</td>
<td>Mode for st command. May also be 1 ('l') or -1 ('b').</td>
</tr>
<tr>
<td>_SHOW_STACK_FLAGS</td>
<td>_showStackFlags</td>
<td>ssParams</td>
<td>Any OR'd combination of: _ssParams (show call parms), _ssLocals (show block locals), _ssCtx (show current execution address of stack frame).</td>
</tr>
</tbody>
</table>

continued
### DB Runtime Routines

The following routines are available in the DB runtime.

As in the C runtime:

- `memcpy`
- `printf`
- `putchar`
- `sprintf`
- `strlen`

As in `CLSMGR.H`:

- `ObjectCall`
- `ObjectCallAncestor`
- `ObjectSend`
- `ObjectSendUpdate`
- `ObjectPost`
- `ObjectPostAsync`

Returns the debug flags of set `<set>`.

```c
_HEX_LONG _GetFlags(U16 set)
```

Sets the debug flags `<set>` to `<flags>`.

```c
void _SetFlags(U16 set, HEX_LONG flags)
```

Delays DB execution for `<ms>` milliseconds.

```c
void _Delay(U32 ms)
```

Sets the systick rate, in milliseconds.

```c
void _SetSystick(U32 ms)
```

Does a `printf` and then raises a DB error, typically forcing DB to command mode.

```c
void _Err(CHAR * pFmt, ...)
```
Allocate globally accessible memory.

```c
void * _Alc(SIZEOF size)
```

Free globally accessible memory.

```c
void _Free(void * pMem)
```

## The ON Command

The `on` command creates a handler for specified events.

In general, the syntax for the `on` command is:

```
[taskSet] on event [eventAction]
```

* `taskSet` A task set as described in Section 7.2.7. Defaults to all tasks.

* `eventAction` C code to be executed when the breakpoint is hit. The default action is to break to the debugger.

There are several categories of events.

### Program Events

An `installBegun` event happens when instance 0 of a program begins execution.

```
[taskSet] on installBegun defNameString [eventAction]
```

An `installDone` event happens when instance 0 of a DLL finishes executing `DLLInit()`.

```
[taskSet] on installDone defNameString [eventAction]
```

A `start` event occurs when any instance of a program besides instance 0 begins execution.

```
[taskSet] on start defNameString [eventAction]
```

### Access Events

An access event occurs when the variable specified by the lvalue is read or written.

```
[taskSet] on access lvalue [eventAction]
```

A store event occurs when the variable specified by the lvalue is written.

```
[taskSet] on store lvalue [eventAction]
```

### Task Events

A tick event happens every systick interval.

```
[taskSet] on tick [eventAction]
```

A step event is an instruction step.

```
[taskSet] on step [eventAction]
```

A pstep event is the same as a step event but skips over CALLs.

```
[taskSet] on pstep [eventAction]
```
A newST event is the creation of a new subtask;

\[\text{[taskSet]} \text{ on newST [eventAction]}\]

A newIST event is the creation of a new interrupt subtask

\[\text{[taskSet]} \text{ on newIST [eventAction]}\]

A terminate event is the termination of a task.

\[\text{[taskSet]} \text{ on terminate [eventAction]}\]

A debReq event happens when a task calls \text{Debugger}().

\[\text{[taskSet]} \text{ on debReq [eventAction]}\]

\section*{Fault Events}

A fault event occurs whenever a hardware fault occurs. The special event fault is shorthand for ORing all of the fault events

\[\text{[taskSet]} \text{ on div [eventAction]}\]
\[\text{[taskSet]} \text{ on nmi [eventAction]}\]
\[\text{[taskSet]} \text{ on into [eventAction]}\]
\[\text{[taskSet]} \text{ on bound [eventAction]}\]
\[\text{[taskSet]} \text{ on invOp [eventAction]}\]
\[\text{[taskSet]} \text{ on extNA [eventAction]}\]
\[\text{[taskSet]} \text{ on invTSS [eventAction]}\]
\[\text{[taskSet]} \text{ on segNP [eventAction]}\]
\[\text{[taskSet]} \text{ on stkOv [eventAction]}\]
\[\text{[taskSet]} \text{ on gp [eventAction]}\]
\[\text{[taskSet]} \text{ on unk [eventAction]}\]
\[\text{[taskSet]} \text{ on fault [eventAction]}\]

\section*{Other Events}

An exit event happens when a block of interpreted C code is exited. Exit events are used to establish cleanup code.

\begin{verbatim}
{   <initialization code>
    !on exit { <cleanup code> } 
    <code that may exit>
 }
\end{verbatim}

An error event happens when an error occurs during execution of interpreted C code.

\begin{verbatim}
{   !on error { printf("BAD ADDRESS\n"); return; } 
    *pData = 0;
 }
\end{verbatim}

An intReq event happens whenever DB’s break key is pressed.

\[\text{on intReq [eventAction]}\]

A bp event occurs whenever the codeAddress is executed.
An oc event occurs whenever a message matching the message pattern is delivered.

The **INSTALL and START Commands**

DB's `install` command does an `OSProgramInstall()` of an .EXE or .DLL.

DB's `start` command does an `OSProgramInstantiate()` of an .EXE or .DLL.

These commands cannot be used to test PenPoint applications because they do not do all of the things that the Application Framework does when installing and instantiating a PenPoint application.

**Context Inside of Breakpoints**

When DB executes code inside of a breakpoint, it does not set the global “current source position” information. (This is done so that it is possible to write breakpoint code that doesn’t disturb that part of the global context.) This means that the source viewing commands (v, u, vu, and uv) may not display what you expect.

To set the current source position when executing code inside a breakpoint, do the following in the breakpoint code:

```
! . ctx
```

Here's a more complete example:

```
> bp SomeFunction {
} ! . ctx;
} ! v a 5;
} ...
}
```

**Cast Operator**

A cast may be used as a postfix operator. For example, `x (int)` is equivalent to `(int)x`.

**Tilde Operator**

The tilde can be used as a postfix dereference operator. For example, `x~` is equivalent to `*x`. 
Chapter 10 / General PenPoint Debugging Techniques

This chapter describes the other tools and techniques that you can use to debug PenPoint programs. Many of these tools or techniques use three facilities provided in the PenPoint™ operating system:

- The DEBUG compiler option, which causes specific message passing macros to send text to the debugger stream.
- Debugger flags, which you can use to turn on and turn off certain behaviors in a program.
- The debugger stream, a pseudo device to which programs can write text. You can view the debugger stream either on a second monitor or in the System Log application, which is described in Chapter 11. After shutting PenPoint down, you can view the debugger stream in a log file, named PENPOINT.LOG.

Unfortunately these topics are tightly interrelated, it is hard to describe one without requiring some knowledge of the other. If you have questions while reading this chapter, keep reading; your question will probably be answered later on.

DEBUG Compiler Option

When you compile an application, you can use the /D compiler switch to create the preprocessor #define name DEBUG.

```
set WCC386=/3s /Oif+ /s /W3 /We /Zc /Zq /fpc /D2 /En /DDEBUG
wcc386p /Foemptyapp.obj emptyapp.c > emptyapp.err
```

PenPoint Uses DEBUG

When you specify the DEBUG #define name in compilation, several PenPoint macros in DEBUG.H, GO.H, and CLSMGR.H are defined so that they send information to the debugger stream (StsWarn, StsFailed, StsRet, StsJmp, and StsOK in GO.H, ASSERT in DEBUG.H, and a number in CLSMGR.H). See the tables at the end of this chapter for a complete listing.

StsWarn evaluates any expression that returns STATUS. If STATUS is an error, StsWarn writes the status value, the file name, and line number to the debugger stream. The message passing macros, StsFailed, StsRet, StsJmp, and StsOK all use the StsWarn macro.

You can use the macro Dbg to comment out single line debug statements when DEBUG is not defined. For example:

```
Dbg(Debugf("Only shows up in DEBUG version"));
```
Often when working on functions called by other functions, you assume that the software is in a certain state. The `ASSERT` macro lets you state these assumptions, and if DEBUG is set, it checks to see that they are in fact the case. If the assumptions are not satisfied, `ASSERT` will send text to the debugger stream. For example, a square root function might rely on never being called with a negative number:

```c
void MySqRoot(int num)
{
    ASSERT(num >= 0, "MySqRoot: input parameter is negative!");
    // Calculate square root...
}
```

If DEBUG is defined and the assertion is not true, `ASSERT` will send “MySqRoot: input parameter is negative!” to the debugger stream.

**Using DEBUG in Your Programs**

You can also test for the DEBUG `#define` name in your programs. If you include `#ifdef` and `#endif` statements to test for DEBUG, you can make your application behave differently when it is compiled with DEBUG. For example:

```c
#ifdef DEBUG
    DPrintf("Pen Moved to location (%ld,%ld)\n", pArgs->xy);
#endif
```

The function `DPrintf()` is described later in this chapter.

**Debug Versions of PenPoint DLLs**

The PenPoint SDK includes debugging versions of the PenPoint DLLs. These print out informative errors and use the DEBUG versions of `STATUS` checking macros.

To use the debugging versions of system DLLs, modify `\PENPOINT\BOOT\BOOT.DLC` to load the debugging versions. For instance, change this line:

```
   go-input-vl(O) \boot\penpoint\boot.dll\input.dll
```

to this:

```
   go-input-vl(O) \boot\penpoint\boot_dll\input.dll
```

**Debugging Flag Sets**

PenPoint sets aside 256 variables, called **debugging flag sets**, that you can use when debugging. Each variable is a 32-bit value, which means that you can assign at least 32-different meanings to each debugging flag set.

Because there are 256 debug flags sets, they can be indexed by an 8-bit character. Commonly, we refer to a specific debugging flag set by the character that indexes that flag. GO has reserved all the uppercase character debug flags sets (A through Z), and has reserved some of the lowercase characters also. To find which debug flags set are available, see the file `\PENPOINT\SDK\INC\DEBUG.H`. 
You can set the value of a debugging flag set and retrieve it. There are several ways to set values in debugging flag sets:

- Using the function `DbgFlagSet()` in a program.
- Using the `debugset` environment variable in PENPOINT\BOOT\ENVIRON.INI.
- Using the `fs` command in either DB or the mini-debugger.

Your application can determine the value of a debugging flag set with the function `DbgFlagGet()`.

Typically, you set a flag value before running a program and your program uses `DbgFlagGet()` to check the value of the flag.

Do not use these in production code; it is fine to use them in DEBUG versions, but they are not administered and could cause hard-to-find errors.

Some PenPoint modules include additional code that can be activated via these debugging flag sets. Header files describe the effects of setting different bits in each debugging flag set. For example, setting the 4000 bit in C causes the debugging version of the Class Manager to print miscellaneous warning messages.

**The Debugger Stream**

As described in the introduction to this chapter, the debugger stream is a pseudo device to which programs can write text.

There are four different ways to view the information sent to the debugger stream:

- If logging is turned on, all information written to the debugger stream is written to the log file.
- You can direct the debugger stream to send its information to a serial port.
- If you have two monitors (VGA and monochrome), the debugger stream is displayed on the second (monochrome) monitor.
- If you have the System Log application running, the debugger stream is captured by the System Log.

These destinations are not mutually exclusive. If all three are active, all three will receive the debugger stream data. The System Log application is described in detail in Chapter 11.

**Configuring the Debugger Stream Destinations**

You can control where the debugger stream is routed by modifying lines in your MIL.INI and ENVIRON.INI files. This section describes the various debugger stream destinations and how you configure them. See the next chapter (Chapter 11) for information on using System Log Application.
Writing to a Log File

To direct the debugger stream to a log file, set the debugger flag D to the value 8000. There are several ways to do this:

- In DB or the mini-debugger, use the fs command: fs D 8000.
- In ENVIRON.INI, add the DebugSet line: DebugSet=/DD8000.

The _ENVIRON.INI file that is shipped with the SDK contains the DebugSet line, but it is commented out. The simplest way to enable logging is to remove the pound sign (#) comment marker at the beginning of the line

```
#DebugSet=/DD8000
```

By default, the debug output is written to the file \PENPOINT.LOG. To change the destination of the log file, add a DebugLog line to your ENVIRON.INI file. For example, these lines cause the debugger stream to be logged in the file MY.LOG in the root of the system volume:

```
DebugSet=/DD8000
DebugLog=\MY.LOG
```

Normally output to the debug log is buffered, to improve performance. However, this can cause problems if the machine crashes before it can write the last buffered data to the log file. To avoid this, you can add a DebugLogFlushCount line to your ENVIRON.INI file. This line specifies the number of characters that can be written to the buffer before the buffer is flushed to the log file.

Writing to a Serial Port

You can direct the debugger stream to write to a serial port. You can then attach a dumb terminal (or a PC running a terminal emulation package) to the serial port and view the debugger stream, much like using a second monitor.

This is most useful when you are debugging a PenPoint program on a PenPoint computer and have no other way to view the debugger stream, other than the PENPOINT.LOG file.

To write debugging information to a serial port, edit your MILINI file and add a SerialDebugPort line to it. The _MILINI file that is shipped with the SDK contains two SerialDebugPort lines already configured, but commented out.

```
# Remove the "#" on the next line to route debugging information to COM1
#SerialDebugPort=1
# Remove the "#" on the next line to route debugging information to COM2
#SerialDebugPort=2
```

Writing to a Second Monitor

Developing PenPoint applications is much more pleasant if you have a two-monitor configuration (one VGA, one monochrome). With two screens you can interact with your program while watching the debugger stream on the monochrome screen. With only one monitor, you can only view the debugger stream when you halt PenPoint (with the Pause key).
If PenPoint is running on a PC and it detects a second (monochrome) monitor, it will display the debugger stream on that monitor. All you have to do is add a monochrome card to your machine and attach a monochrome monitor to the card.

If, for some reason, you want to turn off the debugger stream from the second monitor, add the line `MonoDebug=off` to your MIL.INI file. The _MIL.INI file shipped with the SDK contains a MonoDebug line, but it is commented out.

**Viewing on a Single Monitor**

If you have a single monitor on your PC, you can use the System Log application to view the debugger stream. However, you cannot run the System Log application while PenPoint is booting. To view the debugger stream during the boot sequence, you must:

- Modify the MonoDebug line in your MIL.INI file so that it specifies "off."
- Ensure that your boot volume contains the file `\PENPOINT\BOOT\CONSOLE.DLC`.

The CONSOLE.DLC file lists screen drivers that display debugging information on the VGA screen during booting.

**Writing to the Debugger Stream**

In describing other aspects of debugging, we have already described several ways that you can write to the debugger stream. These tables summarize the functions and macros that send text to the debugger stream.

The functions in the Table 10-1 always write text to the debugger stream (that is, they don't have to be compiled with the DEBUG preprocessor `#define` name).

<table>
<thead>
<tr>
<th>Function</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Debugf</td>
<td>Writes a formatted string (like printf), with a trailing newline.</td>
</tr>
<tr>
<td>DPrintf</td>
<td>Writes a formatted string, like printf, with no trailing newline.</td>
</tr>
</tbody>
</table>

Table 10-2 lists the "warn" form of the PenPoint message passing macros. If you compile these macros with the DEBUG preprocessor `#define` name and the message that they pass returns a status value less than `stsOK`, the macros write a string to the debugger stream. The string contains the name of the file and the line number where the unsuccessful message occurred.

If you compile these macros without DEBUG, they simply call the standard form of the function. For example, without DEBUG `ObjCallWarn` will call `ObjectCall()`.
Table 10-2 lists the warning forms of the PenPoint message passing functions. When the message passed by these functions returns a status value less than \texttt{stsOK}, these functions write text to the debugger stream.

Usually you don’t use these functions but you use the related macros, listed in Table 10-3.

Table 10-3 lists the message passing functions. You can use these macros when invoking a macro or function that evaluates to a status value. If you compile these macros with the \texttt{DEBUG} preprocessor \#define name and their expression evaluates to less that \texttt{stsOK}, the macros write a string to the debugger stream (and some perform further actions). The string contains the
status value and the name of the file and the line number where the unsuccessful expression occurred.

<table>
<thead>
<tr>
<th>Macro</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>StsWarn</td>
<td>If not OK, writes to debugger stream.</td>
</tr>
<tr>
<td>StsFailed</td>
<td>If not OK, calls StsWarn.</td>
</tr>
<tr>
<td>StsRet</td>
<td>If not OK, calls StsWarn and returns with status.</td>
</tr>
<tr>
<td>StsJmp</td>
<td>If not OK, calls StsWarn and jumps to specified location.</td>
</tr>
<tr>
<td>ASSERT</td>
<td>If assertion fails, writes to debugger stream.</td>
</tr>
</tbody>
</table>
Chapter 11 / The System Log Application

The System Log application allows you to view text written to the debugger stream. The System Log application reads (but does not intercept) text written to the debugger stream, and saves the text in its internal buffer.

You can scroll the System Log application's contents so that you can review your debugging session. The System Log application preserves an adjustable number of lines of debugging output. The application also has several menu commands that give you quick access to memory information, enable debug logging, and get and set debug flags.

Loading the System Log Application

The System Log application is a normal PenPoint application. You can configure the PenPoint™ operating system so that the application is loaded at boot time or you can load the application using the application installer.

The line needed to load the System Log application at boot time is already in the SYSAPPINI file. However, the line is commented out. To load the System Log application at boot time, just remove the pound sign (#) comment at the beginning of the line.

Running the System Log Application

The System Log application is a accessory. You can run the application by opening Accessories and then tapping on the System Log icon.
You can scroll, resize, move, open, and close the System Log application as you would any other floating accessory. For more information on using tools, see the end-user documentation.

To close the System Log application, tap its close corner. The lines in the System Log application are preserved (and are also in the debug log file if you enable logging).

**System Log Application Menus**

The System Log application has four menus that you can use to control the application or display system information. They are: Show, Trace, Size, and Font Size.

**Show Menu**

The items in the Show menu direct the System Log application to display information about the current system.

- **Memory Usage** Displays the current memory used by PenPoint and application.
- **Task List** Lists all of the tasks currently running.
- **Device List** Lists all devices currently installed.

**Trace Menu**

The Trace menu allows you to specify the types of messages displayed on the System Log’s window.

- **Application Errors** Display all application and non-system errors from the StdError() function.
Non-error Messages Display all application and system dialog messages from the `StdMsg()` function.

Off Turns off all debugging messages, including those from `Debugf`, `DPrintf`, and `StsWarn`.

The Off choice only appears in the version of the System Log application that was compiled with the `DEBUG` preprocessor `#define` name (in `\_PP\BOOT\APP`). If you can’t see the Off choice, you are using the production version of the System Log application from `\PENPOINT\BOOT\APP`.

Log Size Menu

The log size menu allows you to choose how large a buffer to reserve for the System Log. When the System Log is full, the oldest lines are removed before new lines are added. If you are working with limited amounts of memory, you may want to choose a smaller size for the System Log.

The choices are small, medium, and large. Small is 8K bytes (approximately 6 computer screens of information); medium is 16K bytes; large 32K bytes. The default size is medium.

Font Menu

The font menu allows you to choose the size font to display the system log. Possible sizes are: 8, 10, 12, 14, 18, and 24 point type.
Chapter 12 / PenPoint
Mini-Debugger

In addition to DB, PenPoint has a much simpler kernel debugger. The mini-debugger is not as full featured as DB. It can't display source code lines or symbols, but it has minimal memory requirements, doesn't require any configuration, and can debug any code (including kernel code).

If the mini-debugger is loaded and any process crashes, you can determine what line your program crashed at, what routines called which routines, and what PenPoint modules were involved. All you have to do is log the results of your mini-debugger session, use the mini-debugger to get some system and process information, and compare the results with the .MAP files produced by the compiler and linker.

The mini-debugger lets you:

- Get information about tasks, heaps, segments, memory usage, the memory file system.
- Perform stack traces.
- Disassemble code and display and modify registers.
- Set breakpoints and step through code at the assembler language level.

The mini-debugger is a part of the PenPoint system, so it is always loaded.

## Invoking the Mini-Debugger

When running the PenPoint™ operating system on a PC, pressing [Pause] enters the mini-debugger. Its prompt is a greater-than sign, >. If you're running on a single-monitor configuration, the PenPoint graphical screen is replaced by a text screen, otherwise output appears on the second monitor. You can then type commands to the mini-debugger.

PenPoint also enters the mini-debugger whenever a process has a general protection fault or page fault.

Also a program can explicitly enter the mini-debugger by calling the Debugger() system routine.

## Mini-Debugger and DB

If you have installed DB, then all of the above actions invoke DB instead of the mini-debugger. You can tell the difference between them because the DB > prompt has a "cursor," a dot which indicates the current insertion point. You can enter the mini-debugger from DB by typing mini, and return to DB from the mini-debugger by typing g.
On a PenPoint Computer

When PenPoint enters the mini-debugger, it expects commands from a keyboard. Of course, most PenPoint computers do not normally have a keyboard, so it is difficult to enter a g command to continue.

Therefore, the production version of PenPoint (intended to run on a PenPoint computer) has the D 10000 debugger flag set. This flag disables the mini-debugger. When PenPoint fails, it dumps the registers to the debugger stream and attempts to continue.

Mini-Debugger Commands

Typing ? displays the available mini-debugger commands.

<table>
<thead>
<tr>
<th>Command and Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>?</td>
<td>Evaluate and print expressions.</td>
</tr>
<tr>
<td>ai addr</td>
<td>Display address information.</td>
</tr>
<tr>
<td>ap cmd [cmd]</td>
<td>Execute the commands in all ctx of all processes.</td>
</tr>
<tr>
<td>at cmd [cmd]</td>
<td>Execute the command in all ctx of all tasks.</td>
</tr>
<tr>
<td>bc number</td>
<td>Clear Breakpoint.</td>
</tr>
<tr>
<td>bd { number</td>
<td>*}</td>
</tr>
<tr>
<td>be { number</td>
<td>*}</td>
</tr>
<tr>
<td>bl number</td>
<td>List breakpoints.</td>
</tr>
<tr>
<td>bp number addr</td>
<td>Set breakpoint number at addr.</td>
</tr>
<tr>
<td>cd u32</td>
<td>Call objects with msgDump, pArgs=0.</td>
</tr>
<tr>
<td>cm u32</td>
<td>Convert msg.</td>
</tr>
<tr>
<td>co u32</td>
<td>Convert object.</td>
</tr>
<tr>
<td>cs u32</td>
<td>Convert status.</td>
</tr>
<tr>
<td>ct u32</td>
<td>Toggle trace setting for objects.</td>
</tr>
<tr>
<td>ctx [task-id]</td>
<td>Set the current task.</td>
</tr>
<tr>
<td>d [rangeList]</td>
<td>Display memory at address.</td>
</tr>
<tr>
<td>da [rangeList]</td>
<td>Display ASCII at address.</td>
</tr>
<tr>
<td>db [rangeList]</td>
<td>Display bytes at address.</td>
</tr>
<tr>
<td>dd [rangeList]</td>
<td>Display double words at address.</td>
</tr>
<tr>
<td>dw [rangeList]</td>
<td>Display words at address.</td>
</tr>
<tr>
<td>fd fileName</td>
<td>Display file.</td>
</tr>
<tr>
<td>fl [flag]</td>
<td>Display debug flags.</td>
</tr>
<tr>
<td>fs flag [value]</td>
<td>Set/Clear debug flags.</td>
</tr>
<tr>
<td>g [addr]</td>
<td>Continue program execution (until address).</td>
</tr>
<tr>
<td>hd addr</td>
<td>Display heap summary and allocated blocks information.</td>
</tr>
<tr>
<td>help [cmdName]</td>
<td>Display help.</td>
</tr>
<tr>
<td>hi addr</td>
<td>Heap status.</td>
</tr>
<tr>
<td>hl</td>
<td>Display heaps in process.</td>
</tr>
</tbody>
</table>

...
CHAPTER 12 / PENPOINT MINI-DEBUGGER

Mini-Debugger Commands

Table 12-1 (continued)

<table>
<thead>
<tr>
<th>Command and Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inb port</td>
<td>Input byte from I/O port.</td>
</tr>
<tr>
<td>inw port</td>
<td>Input word from I/O port.</td>
</tr>
<tr>
<td>ldev</td>
<td>Display MIL devices.</td>
</tr>
<tr>
<td>mdb controls</td>
<td>Special Debugger control settings.</td>
</tr>
<tr>
<td>mi</td>
<td>Display task memory information.</td>
</tr>
<tr>
<td>outb port value</td>
<td>Output byte to I/O port.</td>
</tr>
<tr>
<td>outw port value</td>
<td>Output word to I/O port.</td>
</tr>
<tr>
<td>p [physaddr]</td>
<td>Display physical memory.</td>
</tr>
<tr>
<td>q [exitCode]</td>
<td>Exit PenPoint.</td>
</tr>
<tr>
<td>qq</td>
<td>Exit PenPoint, skip standard shutdown.</td>
</tr>
<tr>
<td>r</td>
<td>Display standard registers.</td>
</tr>
<tr>
<td>ra</td>
<td>Display all registers.</td>
</tr>
<tr>
<td>rf</td>
<td>Display floating point registers.</td>
</tr>
<tr>
<td>rx</td>
<td>Display special registers.</td>
</tr>
<tr>
<td>si [semaphore]</td>
<td>Display semaphore information.</td>
</tr>
<tr>
<td>st [addr [n]]</td>
<td>Stack trace at address for n frames.</td>
</tr>
<tr>
<td>t [repeatCount]</td>
<td>Trace—single step.</td>
</tr>
<tr>
<td>ti</td>
<td>Task information.</td>
</tr>
<tr>
<td>tl [taskList]</td>
<td>Display task list.</td>
</tr>
<tr>
<td>tt task</td>
<td>Terminate this task.</td>
</tr>
<tr>
<td>tst taskId [addr]</td>
<td>Stack Trace of another task.</td>
</tr>
<tr>
<td>u [rangeList]</td>
<td>Disassemble code.</td>
</tr>
<tr>
<td>w</td>
<td>Warm boot (PC only).</td>
</tr>
</tbody>
</table>

**Setting Debug Flags**

You use the `fs` command to set debug flags. There are two forms of the command:

```
fs c hhhhhhhhh
```

where `c` is any one byte ASCII character that identifies the debug flag set, and `hhhhhhhh` is a hexadecimal value of 1 to 8 digits. There is no space between the character and hexadecimal value.

```
fs hh hhhhhhhhh
```

where `hh` is a one or two digit hexadecimal value that specifies the debug flag set, and `hhhhhhhh` is a hexadecimal value of 1 to 8 digits. In this form of the `fs` command, spaces are significant. This form allows you to set or clear debug flags that are identified by non-printing ASCII characters (such as those in the ranges 0 through 32 and 128 through 255).
To clear a debugging flag set, set the flag set to 0.

<table>
<thead>
<tr>
<th>Command</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>fs c hhhhhhhh</code></td>
<td>Set flag identified by character c to hex value hhhhhhhhh.</td>
</tr>
<tr>
<td><code>fs c 0</code></td>
<td>Clear flag identified by character c.</td>
</tr>
<tr>
<td><code>fs hh hhhhhhh</code></td>
<td>Set flag identified by hex value hh to hex value hhhhhhhhh.</td>
</tr>
<tr>
<td><code>fs hh 0</code></td>
<td>Clear flag identified by hex value hh.</td>
</tr>
<tr>
<td><code>fl hh</code></td>
<td>Display flag identified by hex value hh.</td>
</tr>
</tbody>
</table>

For example, either of these two commands sets the F debug flag to 10:

```
fs F 0010
fs 46 0010
```

**Using the Mini-Debugger**

If you enter the mini-debugger, you usually:

- Type `fs D 8000` to enable logging (if logging is not already enabled).
- Type `st` to get a stack trace. This should indicate which code the error occurred, and where it was called from.
- If you're familiar with assembly code, type `r` to display registers and `u` to unassemble the code which failed.
- Type `ti` to see the state of tasks.
- Then type `g` to try to continue.

**Map Files**

A map file records line numbers and symbol addresses created by the compiler and linker. A map file is useful when decoding faults and interpreting stack traces produced by PenPoint's mini-debugger. The WATCOM Make files in the `\PENPOINT\SDK\SAMPLE` directories assign the extension `.MPE` to map files for executable modules, and `.MPD` to map files for DLL modules.

To display symbolic names for routines from the mini-debugger, compile your application with the `/En` switch.

**Exception Handling**

If PenPoint entered the mini-debugger due to a general protection fault and you enter `g` to continue execution, the kernel general protection fault handler terminates the task that encountered the fault.

**Understanding Interrupts**

The best place to start interpreting interrupts is with Intel's 386SX/DX/SL Microprocessor Programmer's Reference Manual. In Chapter 9, Exceptions and Interrupts, there's an explanation of the interrupts (Section 9.9) and the `errorCode` registers (Section 9.8).
Some of the interrupts are faults that cause PenPoint to crash. For example, fault 12 is a stack fault. Some faults can be handled by interrupt handlers.

Sometimes an interrupt handler will drop to DB. If this happens, the registers printed by the mini-debugger are the registers for the interrupt handler. However, because this isn’t very useful, DB tries to be helpful by printing the registers for the task that was active when the interrupt handler kicked in. You need to look at those registers. For example:

**FAULT 12 in nonpreemptable task 558**

```
fault 12, errorCode 0000, task 0558
ax 0000FF08 bx 00436C8C cx 0000000F dx 0000003CE si 40774988 di 4031F17E
sp 0044DF20 bp 00436C8C ip 40774CFD 0---0--I--Z--P--
ax 0000FF08 bx 00436C8C cx 0000000F dx 0000003CE si 40774988 di 4031F17E
sp 0044DF20 bp 00436C8C ip 40774CFD 0---0--I--Z--P--
EAX=40670000 EBX=00000000 ECX=00000000 EDX=00000000 EDI=00000000
```

The key set of registers is the first set (lowercase two-letter registers). The capitalized registers are the interrupt subtask calling the debugger. The giveaway is that the task where the fault happened is 558 (a fairly large task ID, so it is probably an application task), but the TSS of the second set is low (probably a system task). The task name (TNAME) is Stak, which makes sense because fault 12 is (from the Intel book) “Stack Exception.”

If you immediately do a stack trace or unassemble the code around the instruction pointer, you get the second set:

```
> u eip-20
40663C65 85DB TEST EBX,EBX
40663C67 740E JE 40663C77
40663C69 68D82B6640 PUSH 40662BD8
40663C6E FF15COD16640 CALL [4066D1C0]
40663C74 83C404 ADD ESP,04
40663C77 66C705306467400100 MOV [40676430],0001
40663C80 E871F89C9F CALL OSDebugger (E00334F6) <<<<
40663C85 66C70530646740000 MOV [40676430],0000
40663C8E 6685DB TEST BX,BX
40663C90 6685DB TEST BX,BX
40663C92 740E JE 40663CA1
40663C93 6812486640 PUSH 40662B14
40663C98 FF15COD16640 CALL [4066D1C0]
40663C9E 83C404 ADD ESP,04
40663CA1 E843FFFFFF CALL DoReinstallDB (40663BE9)
40663CA6 6685DB TEST BX,BX
40663CA9 740E JE 40663CB9
```

All this tells you is that the interrupt subtask dropped into the debugger. What we really want is what was happening that caused the interrupt subtask to mess up, so let’s take a look at the whole task list:
<table>
<thead>
<tr>
<th>Task Id</th>
<th>type</th>
<th>parent</th>
<th>state</th>
<th>class</th>
<th>pri</th>
<th>memused</th>
<th>name</th>
<th>time(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>48</td>
<td>process</td>
<td></td>
<td>ready</td>
<td>low</td>
<td>50</td>
<td>172032</td>
<td>Idle</td>
<td>20.053</td>
</tr>
<tr>
<td>68</td>
<td>Interrupt</td>
<td>48</td>
<td>ready</td>
<td>low</td>
<td>50</td>
<td>172032</td>
<td>Dbl</td>
<td>0.000</td>
</tr>
<tr>
<td>78</td>
<td>Interrupt</td>
<td>48</td>
<td>ready</td>
<td>low</td>
<td>50</td>
<td>172032</td>
<td>TSS</td>
<td>0.000</td>
</tr>
<tr>
<td>88*</td>
<td>Interrupt</td>
<td>48</td>
<td>ready</td>
<td>low</td>
<td>50</td>
<td>172032</td>
<td>Stak</td>
<td>0.000</td>
</tr>
<tr>
<td>98</td>
<td>process</td>
<td>msg wait</td>
<td>system</td>
<td>9</td>
<td>8192</td>
<td>V86x</td>
<td>Scav</td>
<td>0.548</td>
</tr>
<tr>
<td>A8</td>
<td>process</td>
<td>msg wait</td>
<td>high</td>
<td>44</td>
<td>4096</td>
<td>V86x</td>
<td>V86x</td>
<td>40.474</td>
</tr>
<tr>
<td>B8</td>
<td>Interrupt</td>
<td>48</td>
<td>ready</td>
<td>low</td>
<td>50</td>
<td>172032</td>
<td>V86a</td>
<td>0.000</td>
</tr>
<tr>
<td>C8</td>
<td>Subtask</td>
<td>48</td>
<td>ready</td>
<td>med</td>
<td>high</td>
<td>48</td>
<td>Page</td>
<td>0.000</td>
</tr>
<tr>
<td>D8</td>
<td>Subtask</td>
<td>48</td>
<td>msg wait</td>
<td>high</td>
<td>44</td>
<td>4096</td>
<td>Powr</td>
<td>0.145</td>
</tr>
<tr>
<td>E8</td>
<td>Subtask</td>
<td>48</td>
<td>ready</td>
<td>high</td>
<td>44</td>
<td>4096</td>
<td>DmIM</td>
<td>0.070</td>
</tr>
<tr>
<td>F8</td>
<td>Subtask</td>
<td>48</td>
<td>sema wait</td>
<td>med</td>
<td>low</td>
<td>7</td>
<td>Syst</td>
<td>0.000</td>
</tr>
<tr>
<td>108</td>
<td>process</td>
<td>msg wait</td>
<td>med</td>
<td>low</td>
<td>7</td>
<td>835584</td>
<td>Syst</td>
<td>8.711</td>
</tr>
<tr>
<td>118</td>
<td>Subtask</td>
<td>108</td>
<td>msg wait</td>
<td>med</td>
<td>low</td>
<td>7</td>
<td>Timr</td>
<td>0.023</td>
</tr>
<tr>
<td>128</td>
<td>Subtask</td>
<td>108</td>
<td>msg wait</td>
<td>med</td>
<td>low</td>
<td>7</td>
<td>DBm</td>
<td>21.608</td>
</tr>
<tr>
<td>138</td>
<td>Subtask</td>
<td>108</td>
<td>ready</td>
<td>high</td>
<td>45</td>
<td>4096</td>
<td>DBk</td>
<td>0.008</td>
</tr>
<tr>
<td>148</td>
<td>Subtask</td>
<td>108</td>
<td>msg wait</td>
<td>med</td>
<td>low</td>
<td>7</td>
<td>IMgr</td>
<td>0.017</td>
</tr>
<tr>
<td>158</td>
<td>Subtask</td>
<td>108</td>
<td>msg wait</td>
<td>med</td>
<td>low</td>
<td>7</td>
<td>IDSP</td>
<td>0.771</td>
</tr>
<tr>
<td>168</td>
<td>Subtask</td>
<td>108</td>
<td>ready</td>
<td>high</td>
<td>45</td>
<td>4096</td>
<td>Pen</td>
<td>1.723</td>
</tr>
<tr>
<td>178</td>
<td>Subtask</td>
<td>108</td>
<td>ready</td>
<td>high</td>
<td>45</td>
<td>4096</td>
<td>Key</td>
<td>0.001</td>
</tr>
<tr>
<td>188</td>
<td>Subtask</td>
<td>108</td>
<td>ready</td>
<td>high</td>
<td>45</td>
<td>4096</td>
<td>DmKK</td>
<td>0.059</td>
</tr>
<tr>
<td>1A8</td>
<td>Subtask</td>
<td>108</td>
<td>ready</td>
<td>low</td>
<td>30</td>
<td>4096</td>
<td>FSUI</td>
<td>0.165</td>
</tr>
<tr>
<td>1C8</td>
<td>process</td>
<td>msg wait</td>
<td>med</td>
<td>low</td>
<td>7</td>
<td>4096#</td>
<td>SEC0</td>
<td>0.121</td>
</tr>
<tr>
<td>1E8</td>
<td>process</td>
<td>msg wait</td>
<td>med</td>
<td>low</td>
<td>7</td>
<td>4096#</td>
<td>nbt0</td>
<td>0.088</td>
</tr>
<tr>
<td>208</td>
<td>process</td>
<td>msg wait</td>
<td>med</td>
<td>low</td>
<td>7</td>
<td>4096#</td>
<td>NBA0</td>
<td>0.089</td>
</tr>
<tr>
<td>228</td>
<td>process</td>
<td>msg wait</td>
<td>med</td>
<td>low</td>
<td>7</td>
<td>0#</td>
<td>note</td>
<td>0.031</td>
</tr>
<tr>
<td>238</td>
<td>process</td>
<td>msg wait</td>
<td>med</td>
<td>low</td>
<td>7</td>
<td>4096#</td>
<td>Min0</td>
<td>0.414</td>
</tr>
<tr>
<td>258</td>
<td>process</td>
<td>msg wait</td>
<td>med</td>
<td>low</td>
<td>7</td>
<td>28672</td>
<td>MIN0</td>
<td>0.264</td>
</tr>
<tr>
<td>278</td>
<td>process</td>
<td>msg wait</td>
<td>med</td>
<td>low</td>
<td>7</td>
<td>4096#</td>
<td>SET0</td>
<td>0.592</td>
</tr>
<tr>
<td>288</td>
<td>process</td>
<td>msg wait</td>
<td>med</td>
<td>low</td>
<td>7</td>
<td>4096#</td>
<td>HELO</td>
<td>0.082</td>
</tr>
<tr>
<td>2D8</td>
<td>process</td>
<td>msg wait</td>
<td>med</td>
<td>low</td>
<td>7</td>
<td>4096#</td>
<td>INBO</td>
<td>0.086</td>
</tr>
<tr>
<td>2F8</td>
<td>process</td>
<td>msg wait</td>
<td>med</td>
<td>low</td>
<td>7</td>
<td>4096#</td>
<td>OBOO</td>
<td>0.087</td>
</tr>
<tr>
<td>318</td>
<td>process</td>
<td>msg wait</td>
<td>med</td>
<td>low</td>
<td>7</td>
<td>4096#</td>
<td>sta0</td>
<td>0.086</td>
</tr>
<tr>
<td>338</td>
<td>process</td>
<td>msg wait</td>
<td>med</td>
<td>low</td>
<td>7</td>
<td>4096#</td>
<td>CNCO</td>
<td>0.125</td>
</tr>
<tr>
<td>358</td>
<td>process</td>
<td>msg wait</td>
<td>med</td>
<td>low</td>
<td>7</td>
<td>4096#</td>
<td>VKE0</td>
<td>0.114</td>
</tr>
<tr>
<td>378</td>
<td>process</td>
<td>msg wait</td>
<td>med</td>
<td>low</td>
<td>7</td>
<td>4096#</td>
<td>MAS0</td>
<td>0.100</td>
</tr>
<tr>
<td>398</td>
<td>process</td>
<td>msg wait</td>
<td>med</td>
<td>low</td>
<td>7</td>
<td>4096#</td>
<td>ACC0</td>
<td>0.098</td>
</tr>
<tr>
<td>3B8</td>
<td>process</td>
<td>msg wait</td>
<td>med</td>
<td>low</td>
<td>7</td>
<td>4096#</td>
<td>CLOO</td>
<td>0.119</td>
</tr>
<tr>
<td>3D8</td>
<td>process</td>
<td>msg wait</td>
<td>med</td>
<td>low</td>
<td>7</td>
<td>4096#</td>
<td>BO00</td>
<td>0.103</td>
</tr>
<tr>
<td>3F8</td>
<td>process</td>
<td>msg wait</td>
<td>med</td>
<td>low</td>
<td>7</td>
<td>0#</td>
<td>anim</td>
<td>0.007</td>
</tr>
<tr>
<td>408</td>
<td>process</td>
<td>msg wait</td>
<td>med</td>
<td>low</td>
<td>7</td>
<td>0#</td>
<td>hwle</td>
<td>0.012</td>
</tr>
<tr>
<td>418</td>
<td>process</td>
<td>msg wait</td>
<td>med</td>
<td>low</td>
<td>7</td>
<td>0#</td>
<td>hwge</td>
<td>0.010</td>
</tr>
<tr>
<td>428</td>
<td>process</td>
<td>msg wait</td>
<td>med</td>
<td>low</td>
<td>7</td>
<td>290816</td>
<td>ctsb</td>
<td>0.252</td>
</tr>
<tr>
<td>448</td>
<td>process</td>
<td>msg wait</td>
<td>med</td>
<td>low</td>
<td>7</td>
<td>4096#</td>
<td>flap</td>
<td>0.028</td>
</tr>
<tr>
<td>468</td>
<td>process</td>
<td>msg wait</td>
<td>med</td>
<td>low</td>
<td>7</td>
<td>4096#</td>
<td>SIO</td>
<td>0.029</td>
</tr>
<tr>
<td>488</td>
<td>process</td>
<td>msg wait</td>
<td>med</td>
<td>low</td>
<td>7</td>
<td>4096#</td>
<td>HSLI</td>
<td>0.021</td>
</tr>
<tr>
<td>4A8</td>
<td>process</td>
<td>msg wait</td>
<td>med</td>
<td>low</td>
<td>7</td>
<td>20480</td>
<td>atp</td>
<td>0.064</td>
</tr>
<tr>
<td>4B8</td>
<td>Subtask</td>
<td>4A8</td>
<td>ready</td>
<td>low</td>
<td>7</td>
<td>4096#</td>
<td>ATP</td>
<td>0.274</td>
</tr>
<tr>
<td>4D8</td>
<td>process</td>
<td>msg wait</td>
<td>med</td>
<td>low</td>
<td>7</td>
<td>28672</td>
<td>MAR0</td>
<td>0.496</td>
</tr>
<tr>
<td>4E8</td>
<td>process</td>
<td>msg wait</td>
<td>med</td>
<td>low</td>
<td>7</td>
<td>53248</td>
<td>BO01</td>
<td>1.456</td>
</tr>
<tr>
<td>4F8</td>
<td>process</td>
<td>msg wait</td>
<td>med</td>
<td>low</td>
<td>7</td>
<td>32768</td>
<td>NEA1</td>
<td>1.204</td>
</tr>
<tr>
<td>508</td>
<td>process</td>
<td>msg wait</td>
<td>med</td>
<td>low</td>
<td>7</td>
<td>57344</td>
<td>nbt1</td>
<td>2.224</td>
</tr>
<tr>
<td>548</td>
<td>process</td>
<td>msg wait</td>
<td>med</td>
<td>low</td>
<td>7</td>
<td>135168</td>
<td>MAR2</td>
<td>2.357</td>
</tr>
<tr>
<td>558</td>
<td>Subtask</td>
<td>548</td>
<td>ready</td>
<td>low</td>
<td>7</td>
<td>28672</td>
<td>MAR0</td>
<td>0.931</td>
</tr>
</tbody>
</table>

Memory allocated by the system: 3977216
Memory allocated by the loader: 2654208
There's task 558, a subtask our application created. You can then switch context to the task and get a stack trace and look at the instruction pointer.

```
> ctx 558
> u eip-20
40774CDD 4D      DEC EBP
40774CDE 7740     JA  40774D20
40774CE0 55      PUSH EBP
40774CE1 8BEC     MOV EBP,ESP
40774CE3 57      PUSH EDI
40774CE4 56      PUSH ESI
40774CE5 53      PUSH EBX
40774CE6 8B6D08   MOV EBP, [EBP+8]
40774CE9 3EB7D04   MOV EDI, DS:[EBP+4]
40774CF0 66BACE03  MOV DX, 03CE
40774CF5 66EF     OUT DX, AX
40774CF7 66B80503  MOV AX, 0305
40774CFB 66EF     OUT DX, AX
40774CFD 66B451A   MOV AX, [EBP+1A]
40774D01 6683F801  CMP AX, 01
```
TSS  Interrupt task that handles invalid task segment structures. This is Interrupt 10—Invalid TSS from the Intel book. It should never happen.

Stak  Interrupt task that handles stack faults in ring 0. This is Interrupt 12—Stack Exception from the Intel book. Note that most code is in ring 3, and exceeding stack in ring 3 is handled in-line by growing the stack without the kernel doing anything special.

Scav  Process that scavenges, activated by Class Manager.

V86x  Virtual 8086 task. Calls DOS Int 13 functions in real mode, to avoid problems with trying to call these functions in protected mode. Note that tablet hardware doesn't have this task; it can use the MIL to talk to devices in protected mode.

V86a  Handles interrupts in virtual 8086 mode.

Page  Page daemon that writes out dirty pages. If you change data, or memory-map a file and change something in memory, the contents of memory are dirty and have to be written out (a basic virtual memory idea). This process does it periodically.

Powr  Power management, turns off power, handles the physical power button.

DmIM  “Dm” indicates a continuous Driver interface to the MIL. The last two letters represent the logical device (A is 0, so I is logical device 8) and the function (A is 0, so M is function 12). You can enter ldev in the mini-debugger to get a list of logical devices, then look in the DV*.H header to figure out its function.

Syst  Loads DLL code that doesn't have a process. It is the owner for the resources allocated by these DLLs. Observe how much memory it uses (835584 bytes!). The DLLs mentioned in BOOT.DLC are all owned by this task.

Timr  Keeps track of timer events. You can’t send a message during a timer interrupt. Timer interrupts use a semaphore, which is checked by this subtask. When the semaphore is clear, the subtask starts timer processing and message sending.

DBm  The main DB task.

DBk  The DB keyboard reader.

IMgr  The Input manager that grabs pen and keyboard events.

IDSP  The Input dispatcher that dispatches input to windows.

Pen  The pen subtask.

Key  The keyboard subtask.

DmKK  Another continuous Driver interface to the MIL.
FSUI  Handles the file system user interface.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1C8</td>
<td>SECO</td>
<td>0.121</td>
<td></td>
<td>1E8</td>
<td>nbt0</td>
<td>0.088</td>
<td></td>
<td>208</td>
<td>NBA0</td>
<td>0.089</td>
<td></td>
<td>228</td>
<td>note</td>
<td>0.031</td>
<td></td>
</tr>
<tr>
<td>238</td>
<td>Min0</td>
<td>0.414</td>
<td></td>
<td>258</td>
<td>MIN0</td>
<td>0.264</td>
<td></td>
<td>278</td>
<td>SET0</td>
<td>0.592</td>
<td></td>
<td>288</td>
<td>HELO</td>
<td>0.082</td>
<td></td>
</tr>
<tr>
<td>208</td>
<td>INBO</td>
<td>0.086</td>
<td></td>
<td>2F8</td>
<td>OBO0</td>
<td>0.087</td>
<td></td>
<td>318</td>
<td>sta0</td>
<td>0.086</td>
<td></td>
<td>338</td>
<td>CNCO</td>
<td>0.125</td>
<td></td>
</tr>
<tr>
<td>358</td>
<td>VKE0</td>
<td>0.114</td>
<td></td>
<td>378</td>
<td>MAS0</td>
<td>0.100</td>
<td></td>
<td>398</td>
<td>ACC0</td>
<td>0.098</td>
<td></td>
<td>388</td>
<td>CLCO</td>
<td>0.119</td>
<td></td>
</tr>
<tr>
<td>308</td>
<td>BOO0</td>
<td>0.103</td>
<td></td>
<td>3F8</td>
<td>anim</td>
<td>0.007</td>
<td></td>
<td>408</td>
<td>hwle</td>
<td>0.012</td>
<td></td>
<td>418</td>
<td>hwge</td>
<td>0.010</td>
<td></td>
</tr>
<tr>
<td>428</td>
<td>ctsh</td>
<td>0.252</td>
<td></td>
<td>448</td>
<td>flap</td>
<td>0.028</td>
<td></td>
<td>468</td>
<td>SIO</td>
<td>0.029</td>
<td></td>
<td>488</td>
<td>HSLI</td>
<td>0.021</td>
<td></td>
</tr>
<tr>
<td>4A8</td>
<td>atp</td>
<td>0.064</td>
<td></td>
<td>4B8</td>
<td>ATP</td>
<td>0.024</td>
<td></td>
<td>488</td>
<td>Subtask</td>
<td>4A8</td>
<td>ready</td>
<td>medlow</td>
<td>7</td>
<td>28672</td>
<td>MAR0</td>
</tr>
</tbody>
</table>

Now we’re getting into processes. The number on the end of the process name represents the processCount (the same thing that an application receives in main). Therefore, task names that end in 0 are the the application monitor objects created in AppMonitorMain when the applications were installed. This is also usually where the application creates its application class. There are a lot of these process 0 tasks; to save memory PenPoint compacts them. This is what the "#" means in the memory column.

**SECO**  This is process 0 of the section application.

**nbt0**  Process 0 of the Notebook TOC app class.

**anim**  This is a DLL that has a DLLMain. It’s getting loaded after other DLLs because it isn’t a system DLL in BOOT.DLC, it’s an optional DLL. Any app that wants to use clsAnimator has to mention its DLL in the application’s .DLC file, hence the other term for these DLLs, “distributed DLLs.” PenPoint is smart enough to only load a DLL if it is not already loaded (assuming the loaded version matches).

**hwle**  Another non-system, “distributed” DLL, the letter practice DLL.

**hwge**  Ditto, the handwriting gesture practice DLL.

**flap**  The low-level LocalTalk protocol.

**SIO**  The serial I/O DLL.

**atp**  AppleTalk Transport Protocol DLL.
Now we're getting into non-zero application processes, which is where actual application objects are created (in `AppMain()`). In other words, these are processes associated with active documents.

**BOO1** This is the bookshelf application, which is the first time a bookshelf document is activated. It's not surprising that this is the first document actually created; PenPoint creates the bookshelf from the line in `ENVIRON.INI`:

```plaintext```
StartApp = \boot\penpoint\boot\app\bookshelf
```

**MAR2** An application process `Count 2`. In other words, this is the second time a document of this app has been activated. Note that it could be the same document both times—when you turn away from a document the app object may be terminated, which leads to the process being terminated.

**blank** A subtask created by the application task (548).
Part 3 / Tools
List of Figures

16-1 PenPoint Bitmap Editor 167
16-2 Exporting a Bitmap 170
18-1 The Character Selection Window 182
18-2 The Outline Editing Window 183
18-3 View Preference Dialog Box 185
18-4 Smoothing Lines 186
18-5 Merging Shapes 189
18-6 Winding Direction and Merging 190
18-7 Hint Editing Window 191
18-8 Bitmap Edit Window 194
18-9 Font Header Window 196
18-10 Font Attribute Window 197
18-11 Test Sample Dialog Box 198
18-12 Save Font Dialog Box 199

List of Tables

14-1 DOS File System Utilities 161
18-1 Font Header Window Fields 197
18-2 Font Attribute Window Fields 198
18-3 Adobe Font Fields 200
18-4 Mapping of Adobe Standard Encoding to AFII Codes 200
Chapter 13 / Introduction

Part 3: Tools describes the development tools provided with the PenPoint™ operating system SDK.

Some of the tools you use in DOS while developing your application; others are PenPoint tools that you use to refine or document your application.

Organization of Part 3

Chapter 14, DOS File System Utilities, describes DOS programs that you use to view or modify PenPoint-specific file system information.

Chapter 15, Other DOS Utilities, describes DOS programs that you use to compile method tables, compile and manipulate resource files, and eliminate unneeded clutter from Rich Text Format (RTF) Help notebook files.

Chapter 16, PenPoint Bitmap Editor, describes the PenPoint application that you can use to edit bitmaps, such as PenPoint application icons.

Chapter 17, S-Shot Screen Capture Utility, describes the PenPoint application that you can use to gather TIFF images of the screen. You can use these TIFF images in the documentation for your application (the PenPoint user's guides are full of S-Shot images).

Chapter 18, Font Editor, describes a font editor that you can use to create new fonts or edit existing fonts for the PenPoint operating system.
Chapter 14 / DOS File System Utilities

The PenPoint™ operating system's file system is designed to be layered on top of any existing file system. Currently, the only file system used by PenPoint is the DOS file system.

To support the PenPoint extensions to the DOS file system, PenPoint creates a PENPOINT.DIR file in each directory that contains PenPoint files. The PENPOINT.DIR file contains information for each of the PenPoint files in that directory.

In the course of application development, you need to create or modify PENPOINT.DIR entries for some of your DOS files. At other times, you may need to examine the contents of the PENPOINT.DIR file.

The PenPoint SDK includes a set of DOS programs that let you create and modify PENPOINT.DIR files from a PC. These programs are in \PENPOINT\SDK\UTIL\DOS, summarized in Table 14-1 and described in detail later in this chapter.

<table>
<thead>
<tr>
<th>Program Name</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>STAMP.EXE</td>
<td>Adds special PenPoint information to a DOS file or directory.</td>
</tr>
<tr>
<td>GDIR.EXE</td>
<td>Lists the PenPoint names and file system attributes for all the files and directories in a DOS directory.</td>
</tr>
<tr>
<td>MAKLABEL.EXE</td>
<td>Computes the correct hexadecimal values for file system attribute labels.</td>
</tr>
<tr>
<td>PAPPEND.EXE</td>
<td>Appends directory entries from one PENPOINT.DIR file to another, without appending duplicate names.</td>
</tr>
<tr>
<td>PDEL.EXE</td>
<td>Deletes specific directory entries from PENPOINT.DIR files.</td>
</tr>
<tr>
<td>PSYNC.EXE</td>
<td>Scans the current directory and removes any entries from PENPOINT.DIR for which there are no corresponding files.</td>
</tr>
</tbody>
</table>

**STAMP**

The STAMP utility associates a long PenPoint operating system file name or attributes with a DOS file or directory by modifying or creating a PENPOINT.DIR file. The syntax for STAMP is:

```
STAMP [path] /G "PenPoint Name" [ /D dos-name] [/A attrlabel value ...]
```

- **path** An optional DOS path to the PenPoint directory where the file or directory exists. If not specified, STAMP tries to stamp the file in the current directory.
- **PenPoint Name** The PenPoint node name. It must be in quotes.
dos-name The DOS file name of the file or directory.

attrlabel The 4-byte hexadecimal value of the file attribute label. The file attribute label type (FS_ATTR_LABEL) is defined in \PENPOINT\SDK\INC\FS.H.

value The attribute value. Its format depends on the label type. If the label type is fsFixAttr or fsFix64Attr, then the attribute requires a single 4-byte or 8-byte hexadecimal number. If the label type is fsStrAttr, the attribute requires a quoted string. If the label type is fsVarAttr, the attribute requires a set of two-digit hexadecimal numbers (for example, 3A CE 45); STAMP calculates the length of the variable attribute.

You can use the MAKLABEL utility to calculate a file system attribute label. Part 6: File System, in the PenPoint Architectural Reference, explains file system attributes in more detail.

The DOS name is optional once a PenPoint name has been associated. If the DOS name is not specified, STAMP assumes you want to add more attributes. It is OK to specify the DOS name as well as the PenPoint name; however, it is not correct to specify a DOS name that doesn’t match the DOS name that the PenPoint name is associated with.

Example

This example associates the PenPoint name “Tic-Tac-Toe” with the directory \PENPOINT\APP\TTT\STATNRY\TTTSTAT1, and assigns the attribute 00800274 (with the value 1) to the file.

C:> stamp \penpoint\app\ttt\statnry /G "Tic-Tac-Toe" /D tttstat1 /A 00800274 1

You can specify more than one attribute in a single command by adding as many /A parameters as you need. For example:

C:> stamp "Foo" /A 0000004F "Stuff" /A 0000005F "More Stuff"

GDIR

The GDIR utility lists PenPoint names and attributes in a specific directory.

GDIR [path]

path An optional DOS path to the directory to list. If not specified, GDIR lists names in the current directory.

If the specified directory contains a PENPOINT.DIR file, this utility lists the PenPoint name, number of attributes, and the attributes for each file and directory listed in PENPOINT.DIR.

MAKLABEL

The MAKLABEL utility constructs a hexadecimal attribute label from a class value and an index value.

MAKLABEL {FIX | F64 | STR | VAR} class index

The attribute 00800274 causes the document to appear in the pop-up stationery menu.
**FIX | F64 | STR | VAR** The first argument specifies the type of file system attribute, corresponding to fsFixAttr, fsFix64Attr, fsStrAttr, and fsVarAttr in `\ PENPOINT\SDK\INC\FS.H`

- **class** The administrated number of the attribute's class, in decimal.
- **index** The attribute's index, in decimal.

**MAKLABEL** generates the hexadecimal file system attribute labels required by STAMP. It has similar parameters as the FSMakeAttr macro in `\ PENPOINT\SDK\INC\FS.H` which are used to define file system attributes in header files.

### Example

`\ PENPOINT\SDK\INC\AUXNBMGR.H` defines the file system attribute, which indicates that stationery should be on the pop-up stationery menu:

```c
// Should a piece of stationery be on the stationery menu?
#define anmAttrStationeryMenu FSMakeFix32Attr(clsAuxNotebookMgr, 1)
typedef enum ANM_ATTR_STATIONERY_MENU {
    anmNotOnMenu = 0,    // Same as no attribute
    anmOnMenu    = 1
} ANM_ATTR_STATIONERY_MENU;
```

`\ PENPOINT\SDK\INC\UID.H` gives the administered number for `clsAuxNotebookMgr`:

```c
#define clsAuxNotebookMgr MakeWKN(314,1,wknGlobal)
```

So, the command to print out the hexadecimal value of `anmAttrStationeryMenu` is:

```
c:> \penpoint\sdk\util\dos\maklabel fix 314 1
```

This gives the correct value for the `attrlabel` parameter to STAMP.

---

**PAPPEND**

The PAPPEND utility appends directory entries from one PENPOINT.DIR file to another. Before appending an entry, PAPPEND makes sure that the entry doesn’t already exist in the target file. If the entry exists already, PAPPEND does not append that entry. PAPPEND also eliminates any empty directory entries from the source while appending.

The syntax for PAPPEND is similar to the standard C function `strcat`:

```
PAPPEND target-file source-file [/G "PenPoint-name"] [/V]
```

- **target-file** The file to which you are appending a directory entry.
- **source-file** The file from which the entries are being copied.
- `/G "PenPoint-name"` Specifies that PAPPEND is to append only the entry with this name to the target file.
- `/V` Directs PAPPEND to tell you what it has done (verbose mode).
Example 14.4.1

When you copy an application directory to \PENPOINT\APP, you need to add the entry for that application directory to PENPOINT.DIR. The easiest way to do this is to append the entry from the application directory’s source disk.

B:\PENPOINT\APP> pappend penpoint.dir e:penpoint.dir /g "DrawingPaper" /v Appended "DrawingPaper"

PDEL 14.5

PDEL deletes a specific entry from the PENPOINT.DIR file in your current directory.

The syntax for PDEL is:

```
PDEL "PenPoint-name" [/B] [/D "directory"] [/V]
```

- **PenPoint-name** Specifies the PenPoint operating system name of the file to delete from PENPOINT.DIR.
- **/B** Directs PDEL to create a backup file named PENPOINT.BAK.
- **/D “directory”** Directs PDEL to find PENPOINT.DIR in the specified directory.
- **/V** Directs PDEL to work in verbose mode.

PSYNC 14.6

PSYNC examines the PENPOINT.DIR file in your current directory and compares the directory entries to the actual files in that directory. When it finds an entry for a file that doesn’t exist, PSYNC removes the entry from PENPOINT.DIR.

The syntax for PSYNC is:

```
PSYNC [/B] [/D "directory"] [/P] [/V]
```

- **/B** Directs PSYNC to create a backup file named PENPOINT.BAK.
- **/D** Directs PSYNC to find PENPOINT.DIR in the specified directory.
- **/P** Directs PSYNC to prompt you before removing a directory entry.
- **/V** Directs PSYNC to work in verbose mode.
Chapter 15 / Other DOS Utilities

In addition to the file system utilities, the PenPoint™ operating system provides a number of other DOS utilities that are used in developing applications and running PenPoint.

**GO Batch File**

\penpoint\sdk\util\dos\go.bat is a batch file that starts up PenPoint on a PC. You can modify this batch file to ensure that PenPoint removes TSRs (terminate and stay resident) before running PenPoint, or to have PenPoint check hard disk consistency after running PenPoint. GO.BAT is explained in more detail in Chapter 3, Running PenPoint on a PC.

**PenPoint Method Table Compiler**

\penpoint\sdk\util\clsmgr\mt.exe is the PenPoint method table compiler. It compiles a method table source file into an object file that you link with your class's code. Its use is explained in the *PenPoint Application Writing Guide* and in Part 1: Class Manager of the *PenPoint Architectural Reference*.

**Resource Utilities**

You can create PenPoint text resource files two ways: programmatically (from within a PenPoint application) or by hand. Usually when you create something like a quick-help resource, you build it by hand, then compile using the PenPoint resource compiler (RC). RC enables you to create or append to PenPoint resource files.

The executable for RC is in \penpoint\sdk\util\rc.exe.

To append resources from one resource file to another, use RESAPPND. RESAPPND also compacts resource files by removing deleted or duplicate resources.

For example, MAKEFILE for the Tic-Tac-Toe sample application uses RESAPPND to append all of its separate resource files to its AIP.RES resource file, thereby creating its application resource file.

RESAPPND is in \penpoint\sdk\util\resappnd.exe

To view the contents of a resource file, you can use RESDUMP. If you want to decode the state of a PenPoint document, you can use PenPoint to copy the document from the TOC to a disk browser, then under DOS use RESDUMP to dump the contents of the document's DOC.RES file.

RESDUMP is in \penpoint\sdk\util\resdump.exe.

For more information on resource files and resource compiling, see Part 11: Resources in the *PenPoint Architectural Reference*. 
Chapter 16 / PenPoint Bitmap Editor

The user interface to the PenPoint™ operating system uses icons to identify items in the Bookshelf, Notebook, cork margin, accessories and other notebooks.

You can use the bitmap editor to create **icons** and small **sampled images** (images made of pixels). The bitmap editor has a fairly comprehensive set of tools for basic editing of small **pixelmaps**: foreground and background colors, image shifting, capture from the screen, and so on.

![PenPoint Bitmap Editor](image)
Most applications use two icons: a large one (32 by 32) and a small one (16 by 16). By default, the small icon is used for items on the Bookshelf and in Accessories (draw a check over any icon on the Bookshelf to change style). The small icon is also displayed next to documents in a table of contents or other browser display.

When an application has bitmaps for these two icons in its resource file, the PenPoint operating system displays them in the appropriate situations. Otherwise, it uses default icons from the system resource file.

**Elements of a Bitmap**

You can use the bitmap editor to access and change three basic bitmap elements:

- The array of pixels that make up the bitmap's image.
- The array of bits that indicates which pixels will be visible when the bitmap is displayed. This array is called the **mask**. Only if a bit in the mask is on will its corresponding pixel be visible when the bitmap is displayed.
- The hot spot, which locates the “origin” of the bitmap. This, for example, is the pixel in the bitmap that will be drawn underneath the pen when the bitmap is being used as a cursor.

The bitmap editor has three modes, one for editing each of the three elements described above: Pixel Paint, Mask Paint, and Hotspot Paint.

In Pixel Paint mode, you can edit the bitmap's pixels. When you drag the pen, the pixels that the pen touches change, depending on the color of the pixel on which you started.

- If you started on a pixel that is not in the current ink color (any of the other three colors), the pen will draw that color.
- If you started on a pixel that is the current ink color, the pen will draw the current background color.

In Mask Paint and Hotspot Paint mode, only the mask or the hot spot are affected by dragging the pen in the bitmap editor.

**Importing a Bitmap**

To import a bitmap from an existing document, use the Connections notebook to find the application's resource file (such as PENPOINT\APP\SomeApp\APP.RES), tap-press to copy it, then drag the copy into the table of contents of the Notebook. A menu of possible import types will appear: choose Bitmap Editor; for possible bitmap types, choose either App or Small App.

A new bitmap editor document will appear containing the bitmap. The document remembers where the bitmap resource came from, and will replace it in the resource file.

For more information on importing documents, see PenPoint's end-user manual, *Using PenPoint*. 
Exporting a Bitmap

As mentioned above, the PenPoint operating system looks in an application's resource file for the application's bitmaps. The application's resource file is the file APPRES in the application's directory. For example, the Tic-Tac-Toe program's application resources, including bitmaps, are in \PENPOINT\APP\TIT\APP.RES. Some programs create a resource file in advance; others only have a resource file after installation under PenPoint.

Once you have created a bitmap for an application, you must export that bitmap to the application's resource file so that PenPoint can access it.

To export a bitmap, tap on the Exporting menu button under Options. The Resource ID field on the Exporting card is a pop-up choice, which allows you to specify which of the two application icon resources use this bitmap (large or small).

The only two icon resources used for application's bitmaps are the first two: App for the 32 by 32 icon and Small App for the 16 by 16 icon. Depending on the user's preference, PenPoint will use one or the other of these icon resources when displaying the application's icon. The other (Custom) resource ID is used by other parts of PenPoint and is usually not relevant for applications.

Once you have chosen which resource to use, apply the change and close the option sheet. You then export the bitmap by turning to the table of contents and moving or copying the bitmap document that you just modified to a disk file. PenPoint tells you that it can export the file and prompts you for the file format. Tap on Resource File.

You can export a bitmap to an existing resource file, or create a new one; if the existing resource file already contains a bitmap, PenPoint asks you if you want to replace the existing bitmap. You can also use the RESAPPND utility to append bitmap resources.

For more information on exporting documents, see Using PenPoint, PenPoint's user manual.
Exporting to Home

When you import a bitmap from an existing resource file, the bitmap editor remembers the path to the resource file from which it came (including the volume name). When you are ready to write the bitmap to its resource file, you can tap on Export to Home in the Document menu.

You can see the path to the bitmap's home in the Exporting option card; however, you cannot modify the path.

Bitmap Editor User Interface

The application menu for the bitmap editor contains menus for Document, Edit, Options, Ink, Back, and Size. This section describes the Edit, Options, Ink, Back, and Size menus.

There is no undo capability in the bitmap editor. However, if you want to save your work, you can always tap Checkpoint in the Edit menu to save the document.
in its current state. If you make an unintended modification, you can then use Revert to return the bitmap to its state when you last used Checkpoint.

The Document Menu

The Document menu contains standard document menu buttons.

If you imported the current bitmap from a resource file, Export to Home also appears in the document menu. This non-standard item allows you to export the bitmap back to the resource file from which it came.

The Edit Menu

You choose an editing mode from the first three entries in the Edit menu:

- **Pixel Paint**  Displays only the bitmap's pixels so you can edit them.
- **Mask Paint**  Displays the bitmap's pixels and its mask. You can only edit the mask (indicated by an “X” over masked pixels). The masked pixels are the only ones that will be painted when the bitmap is displayed.
- **Hotspot Paint**  Displays the bitmap, but only lets you set the hot spot for the bitmap. The hot spot is indicated by an “X” in this mode. Hotspot mode is only of interest for bitmaps that will be used as cursors. (There is currently no way for an application to install a custom cursor.)

The other entries in the Edit menu are:

- **Fill**  Fills the entire bitmap area with black in Pixel Paint and Hotspot Paint. In Mask Paint, fill turns the entire mask on.
- **Erase**  Fills the entire bitmap area with the current background color in Pixel Paint and Hotspot Paint. In Mask Paint, erase turns the entire mask off.
- **Invert**  Inverts the colors in the bitmap in all modes. The bitmap has four possible colors: black, dark gray, light gray, and white. In inversion, black and white are switched and dark gray and light gray are switched.
- **Rotate**  Rotates the bitmap by 90 degrees clockwise.
- **Horizontal Flip**  Flips the bitmap around the y-axis.
- **Vertical Flip**  Flips the bitmap around the x-axis.
- **Generate Mask**  Turns the mask on for each pixel that is not the background color. The mask includes any closed areas in the bitmap (those that are surrounded by pixels that are not the background color).
- **Capture**  Displays a square on the screen that tracks the pen. When the pen is lifted, the area under the square is copied into the bitmap editor.
- **Shift Left, Shift Right, Shift Up, and Shift Down**  Shift the bitmap by one pixel (bit) in the indicated direction. Pixels that are shifted out of the bitmap on one side reappear on the other side (rotational shift).
The Options Menu

The Options menu provides standard access to the Controls, Access, and Comments option cards. The two interesting option cards added by the bitmap editor are:

- **Exporting** Allows you to specify the Resource ID for an application icon.
- **Custom Resource ID** Allows you to specify the Resource ID for a custom icon.

The Exporting Option Card

You use the About Exporting option card to specify how the bitmap editor is to export a bitmap.

- **Resource ID** Allows you to specify the Resource ID for the bitmap. This is a pop-up choice that provides you with two predefined PenPoint Resource IDs (App and Small App) and a Custom Resource ID. Most applications will only use App and Small App. You should set the Resource ID to App when editing a 32 by 32 bitmap, and to Small App when editing a 16 by 16 bitmap.
- **Encoding** Allows you to specify how many bits per pixel to use when exporting the bitmap. To save space, this should be as small as possible. For bitmaps that use 4 gray values, the encoding should be 2 bits per pixel.

The Custom Resource ID Card

If you are creating a bitmap that needs a Resource ID other than the two offered in the Resource ID pop-up choice, you can:

- Choose Custom on the Exporting option card.
- Tap on Apply.
- Turn to the Custom Resource ID option card.

The Custom Resource ID card allows you to specify a Resource ID for the bitmap. Applications can display bitmaps with Custom Resource IDs by using **clsBitmap** or **clsIcon** to read in the Resource ID. For more details, see parts 3 and 4 in the *PenPoint Architectural Reference* and the header files for **clsBitmap** and **clsIcon**.

The three fields on the card specify the Resource ID (as defined in `\PENPOINT\SDK\INC\RESFILE.H`):

- The Resource Class field contains the administered number for the class that defines this Resource ID.
- The Resource Scope field specifies whether the Resource ID is global, process global, or private.
- The ID field contains an ID for this particular bitmap resource.
In your code, you'll probably define the same Resource ID using `MakeWknResId(tag, id)`. *tag* identifies the class field and the scope, which is usually a class UID; *id* is a number between 0 and 255 that identifies the resource for this class.

For example, the Resource ID for the “SmallApp” resource used by PenPoint consists of:

- The resource class for clsApp, which is 13.
- The Resource ID, which is global.
- The ID value, which is 1.

To read this resource from a resource file:

```c
#define clsApp MakeGlobalWKN(13, wknGlobal)

RES_FILE file;
RES_READ_OBJECT read;
OBJECT bitmap;
ObjectCall(msgNewDefaults, clsObject, &read.new);
read.mode = resReadObjectMany;
read.resId = MakeWknResId(clsApp, 1);
ObjectCall(msgResReadObject, file, &read);
bitmap = read.object.uid;
```

Once you have read in the resource, you can use clsIcon or clsBitmap to display the bitmap.

**The Ink Menu**

The Ink menu allows you to set the ink color for painting operations when in Pixel Paint or Hotspot Paint mode.

This setting is ignored in Mask Paint mode.

**The Back Menu**

The Back menu allows you to set the background color for painting operations when in Pixel Paint or Hotspot Paint mode.

This setting is ignored in Mask Paint mode.

**The Size Menu**

You use the Size menu to change the bitmap’s size. The possible sizes are: 8 by 8, 16 by 16, 24 by 16, 32 by 32, 40 by 32, and 48 by 32.

- Use 32 by 32 size for large icons such as the App resource.
- Use 16 by 16 size for small icons such as the Small App resource.
- The 8 by 8 size is used within PenPoint for some cursors.
Chapter 17 / S-Shot Screen Capture Utility

S-Shot is a PenPoint™ operating system application that allows you to capture an image of the PenPoint screen. It saves the image to disk as an uncompressed gray-scale TIFF file, which you can then import into a variety of paint or drawing programs on a PC.

**Using S-Shot**

S-Shot essentially provides a user interface to ImagePoint’s `msgDecScreenShot` message. This captures part of the screen in a TIFF file. To capture any portion of the screen, S-Shot disables window clipping.

One full-screen snapshot takes about 130 kilobytes.

If you are taking screen shots on a PC, you can store screen shots on your PC’s hard disk. On a PenPoint computer, you’ll need to make sure there is room on your hard disk, or some removable media, to store the screen shots.

**Installing S-Shot**

S-Shot is part of the PenPoint SDK; its files are in `\PENPOINT\SDK\APP\SSHOT`. You can include S-Shot in PenPoint running on a PC by adding a line to `\PENPOINT\BOOT\APPINII`.

**Using S-Shot**

S-Shot is an accessory. To use it, open the Accessories notebook and tap on S-Shot.

The S-Shot window allows you to specify:

- The area that will be included in the screen shot.
- The delay before the screen shot is taken.
- The path to the file where S-Shot will store its TIFF image.

**Specifying an Area**

S-Shot can create an image from the entire screen (tap on the Full Screen button) or a specific area of the screen. To specify an area of the screen, you can do one of two things:

- Specify the size x-y and origin x-y location by entering specific values.
- Tap on the Set Area... button, which allows you to select the area visually.
If you tap the Set Area... button, S-Shot clears its window and displays a rectangular box in the center of the screen. This box delineates the area that S-Shot will capture.

You can expand or contract the box by placing the tip of the pen on one of the lines and dragging the line up, down, left, or right.

You can move the box by placing the tip of the pen in the center of the box and dragging the box wherever you want it to go.

**Specifying a Delay**

Obviously, you want the screen to look a certain way when you take the snapshot. Before capturing a screen, you should get its state close to the state you want, but there are two obstacles:

- The S-Shot window is in the way. When you take the snapshot, S-Shot takes down its window, but it takes time for the applications underneath the S-Shot window to repaint. This may take several seconds.
- You want to take a snapshot of a transient visual, such as ink on the screen, or quick help, or a highlighted menu button.

To take a broader range of snapshots, you can set a delay before S-Shot takes its snapshot. The default is five seconds, which should be enough time for the screen to repaint and for you to get the screen set up as you desire.

If you need more time, change the value.

**Specifying a File Name**

The default file name is `\BOOT\FILEA.TIF`. This is appropriate for taking a screen-shot on a PC running PenPoint, because `\BOOT` will be the hard disk containing the `\PENPOINT` hierarchy. However, on a PenPoint computer, you *must* change the file name to another name.

1. Tap on the File Name field to bring up an editing pad. The name you use *must* consist of two backslashes, a volume name, a backslash, and a file name. The volume name should be the DOS volume label of the disk on which you want to create the TIFF file. Use the DOS LABEL command to label a floppy disk or check its label.

The file name could be any valid PenPoint file name. However, since the main use of S-Shot is to create TIFF files for editing on other computers, it's a good idea to adhere to the DOS file-naming conventions (no file name may have more than 8 characters). Leave the extension as `.TIF`. The writing pad restricts characters to uppercase.

2. Tap **OK** to apply your changes and dismiss the editing pad.
Do It

When you are ready to take the snapshot, tap Take Snapshot.

If the volume in S-Shot’s file name is not connected, PenPoint displays a note prompting you to insert that volume.

The S-Shot window disappears from the screen, causing windows underneath it to repaint.

PenPoint will beep every second as it counts down the delay, then will beep a different tone. At this point it captures the screen to a TIFF file on disk. A full-screen snapshot takes about a minute to write to disk. During this time PenPoint will be frozen.

When the snapshot is complete, the S-Shot window reappears.

You will notice that S-Shot increments the snap file name in its window, so that if the file name was \BOOT\FILEA.TIF before, it becomes \BOOT\FILEB.TIF. If this is a reasonable name, you can keep it and take the next snapshot.

Hints on Using S-Shot

Taking Snapshots of Gestures

Set the delay high enough to give you time to do what you want. When you draw the gesture, don’t lift the pen.

Impossible Snapshots

You can’t take a picture of the screen during a page turn, or with the busy clock on the screen.

It’s difficult to take a snapshot of disk operations such as the Disk Manager, import/export, or installation, because S-Shot prompts you to install its own volume, which changes the status of connected volumes.

Full-Screen Snapshots

The GO Computer screen is 640 pixels high by 400 wide.

Using S-Shot Files

On an Apple Macintosh Computer

You can use Apple File Exchange on a Superdrive-equipped Apple Macintosh computer to read from DOS-formatted 3-1/2 inch floppies, or use some cross-platform network, such as SITKA’s TOPS or Novell’s Netware to transfer files.
To read a TIFF file into Macintosh programs such as Aldus FreeHand and Studio/8, you need to change the type of the file to TIFF using a Macintosh utility such as ResEdit, DiskTop, or DiskTools (in advanced mode).

**IBM-Compatible PC**

The files that S-Shot creates can be imported directly into a number of programs that run under MS-DOS, such as Microsoft Word, WordPerfect, Micrografx Designer, and so on.

Just insert the disk into your PC and choose your program's **Import TIFF** option.

**TIFF Problems**

There are many TIFF file variants, and most programs don't read in all flavors. In S-Shot, ImagePoint™ (the PenPoint operating system's imaging software) creates a "standard" uncompressed 4-bit gray-scale TIFF image. If your chosen PC application cannot read it, contact your vendor for assistance.

**Bugs**

S-Shot doesn’t validate the numbers you enter for coordinates, or the name you give for the file.

S-Shot doesn’t tell you when your disk is full, it just creates a small file.
Chapter 18 / Font Editor

The PenPoint font editor, FEDIT, enables you to create and modify outline 8-bit font files for the PenPoint™ operating system.

FEDIT runs under Microsoft Windows, version 3.0 or higher.

With FEDIT you can:
- Create and manipulate outline shapes.
- Create and manipulate hints.
- Integrate bitmaps as a part of a font.
- Read and write fonts in PenPoint font format and in other font formats.

Introduction and Concepts

The 8-bit characters that you create with FEDIT are identified using the AFII (American Society for Font Information Interchange) numbering scheme. While PenPoint 1.0 uses 8-bit characters, future releases of PenPoint will use 16-bit characters. The scheme for identifying 16-bit characters is not available at this time. Be aware that 8-bit characters created in FEDIT will be incompatible with the future 16-bit characters. In all likelihood, GO will not extend FEDIT to support 16-bit characters.

If you use a font that you created with FEDIT for special decorations in your user interface, we recommend that you use icons (which you create in the bitmap editor).

Getting Started

Run the program \PENPOINT\SDK\UTIL\DOS\FEDIT.EXE from within Windows. A blank window will appear. To edit an existing PenPoint font file:


2. Choose a file to open. A PenPoint font file should have the extension .PCK. Depending on the complexity of the font, it will take a moment for FEDIT to initialize the opened font database.

3. Choose the Choose command under the Character menu. A character selection dialog appears. It lists all the characters in the font. You can scroll the character list with the scroll bar below it.

4. Click on the character you wish to edit and click on the OK button. The main window will now repaint and you're ready to edit the shape of this character.
To create a new font file:

1. Choose New under the File menu.

2. Choose New under the Character menu. A dialog box appears. Enter the AFII number and the setwidth of the new character. The AFII number identifies a specific character; the setwidth specifies the amount of space that follows a character.

3. The main window repaints with an empty edit window. You’re ready to add shapes to the new character.

**File Formats**

FEDIT supports several kinds of font files: the Nimbus-Q format, GO’s proprietary packed format, and a limited support of Adobe’s Type I font file. You may specify the input or output format to one of these three with the Options menu.

The extension of a Nimbus-Q file is .FNT; GO’s file extension is .PCK; Adobe’s file extension for Type I font is PFB.

You can easily convert the formats. For example, to convert a Nimbus-Q file to a GO packed file, use the following steps:

1. Choose Nimbus-Q Input under the Options menu.

2. Open the Nimbus-Q file with the Open command under the File menu.

3. Choose GO packed Output under the Options menu.

4. Choose Save As under the File menu. You’ll be prompted for an output file name. The saved file now has the GO packed format.

You can convert a GO packed format to the Nimbus-Q format by slightly reversing the procedure. Note, however, that the Nimbus-Q format does not have provisions for saving any bitmap rendering of the font. Thus, if a GO font contains bitmaps and you save it as a Nimbus-Q font, the bitmaps will be discarded. The same is true of converting a GO packed format to the Adobe Type I font format.

**Technical Notes on Character Composition**

A character is composed of the following elements:

- Its AFII number, which is its identity in the font. FEDIT takes a 16-bit signed representation for this field for ease of use but the sign has no significance.

- A setwidth, which specifies the amount of space to be added after the character has been rendered.

- One or more closed shapes, which define the appearance and the placement of the character. For simplicity, a closed shape will simply be referred to as a shape. Each shape is composed of connected straight lines or Bezier curves. Each line or curve in a shape is called a segment. The lines and curves that
define a shape have direction: clockwise or counter-clockwise. Direction is important when filling and merging.

**Font Units**

The setwidth and the control points that define the segments are in the unit of the integer Cartesian grid of \([-2048, 2047] \times [-2048, 2047]\). They are called font units. To render a font with a particular point size, rotation, or italicization, the setwidth and the control points are transformed by a transformation matrix that contains the scaling, rotation, and so on, with 1000 as the standard scaling factor.

For example, to render a 12-point font on a 300 dpi device, the scaling factor will be \((12 \times 300) / (72 \times 1000)\). Although you’d never need to compute these numbers for FEDIT, you need a rough grasp of sizes and proportions to design a good font. For example, a font whose shapes are all inside the square grid of \([300, 300] \times [300, 300]\) is undesirable because even a large point size will produce only a very small font. Most popular fonts, such as Times Roman and Helvetica, have characters sized around 1000 x 1000.

**Character Placement**

The set of shapes that make up a character implicitly defines a rectangle that minimally bounds all the shapes in that character. This is called, not surprisingly, the bounding rectangle of the character.

How does the bounding rectangle affect the placement of characters next to each other in a string? For simplicity, let us assume that we are rendering a string on a device where one font unit is one device pixel; that is, the transformation matrix is the identity, along the x-axis.

First note the position of the bounding rectangle relative to the x-axis, or the base line. The base line is the common reference line that allows the characters in a string to line up. Thus the base of most upper case letters should be directly above or very slightly below the base line while the tails of characters such as g, p, or q will extend far below the base line.

Recall that the setwidth of character X is the value added to the x-coordinate of the cursor each time the character X is drawn. (The setwidth is quite independent on the character’s bounding rectangle.) The new x-position of the cursor is where we’d place the origin \((0, 0)\) of the next character.

The bounding rectangle of the next character is thus important in determining where it is placed after the setwidth of the previous character has been added to the cursor. If, for example, the origin of the character lies inside its bounding box, its left-most edge will be placed to the left of the next cursor position. It is also possible for the setwidth to be inside the character’s bounding rectangle so that the bounding rectangle of the next character to be drawn in the string can overlap the current character. This is useful for a “pseudo-kerning” effect as illustrated on the right.
You may examine these character metrics in a dialog box with the Info command under the Character menu. In addition, the Info command lists the number of lines and curves in the character. You can also alter the AFII number of the character in this dialog box.

**Control Point Placement**

A Bezier curve is composed of four control points, two of which are the end points of the curve, while the other two lie outside of the curve (called the control points). The bounding rectangle of a character is computed based on the coordinates of the control points. The bounding rectangle is *not* based on the pixels that would be drawn if we render the character.

It is possible (but highly discouraged) for a character to have a bounding rectangle that minimally bounds the control points, but not be minimal with respect to the actual pixels that would be drawn if we render the character. The font subsystem in PenPoint computes character metrics based on the assumption that the bounding rectangle of a character bounds not only control points, but rendered pixels as well.

It is always possible to break up a Bezier curve to satisfy this requirement using the mitosis operator described below.

**Adding and Deleting a Character**

To add a character, choose the New command under the Character menu and specify the new character's AFII number and setwidth in the dialog. Each character in a font must have a unique AFII number. FEDIT will not allow you to assign an AFII that is already used to a new character in the font. Each font file can hold up to a maximum of 512 characters.

To delete a character, choose the Choose command under the Character menu to invoke the character list dialog. Click on the character you wish to delete, then click on the Delete button.

**Editing Character Shapes**

To edit character shapes, you first choose a character from the character selection window. To bring up the character selection window, click on the Choose command from the Character menu. The window shown in Figure 18-1 appears.
• To choose a character for shape or hint editing, click on the character, then click on the OK button (or just double-click on the character).

• To delete a character, click on the character and click on the Delete button.

• To change the AFII number of a character, click on the character. The current AFII number appears on the leftmost box. Type in the new AFII number and click on the Set Glyph button.

Caution None of the operations executed from this window are undoable. Think twice before you delete a character.

The Outline Editing Window

To edit the shapes in a character, click on a character in the character selection window, then click on the Shape command from the Outline menu. The outline editing window will appear. Figure 18-2 shows the outline editing window and identifies some of the terms in the window.

• The edit window is where you manipulate the shapes that make up the character.

• The sample windows show two samples of the rendering of the current shape: one portrait, one landscape (rotated 90 degrees). When you drag the borders of these windows, the point size of the samples change; this allows you to examine the character at different point sizes.

• When you click in either of the sample windows, the sample windows update themselves to reflect the shape of the character in the edit window. The sample windows do not redraw themselves each time you alter the shapes in the edit window.
To zoom into a portion of a character for fine editing, click and drag in the portrait sample window; as you drag, a zoom rectangle will appear. Where you first click is the center of the rectangle. When you release the mouse button, the edit window will show that portion of the character you have just selected. The scale status field shows you the scale factor relative to font units at which the character (or part of it) is rendered in the edit window. Thus, 100% means that one font unit equals one device unit on the screen.

When you drag out a zoom rectangle, holding down the left mouse button adjusts the size of the zoom rectangle. To move the zoom rectangle, hold down both the left and right buttons.

The coordinate values at the top of the edit window show the position of the cursor in font units. Note that the cursor moves in the screen device's resolution, which is usually at a lower resolution than the font units. FEDIT can only scale the font coordinates from device coordinates. These numbers are therefore not exact.

You can pan around the character with the horizontal and vertical scroll bars.

The setwidth line (with a small control box at the base line) shows the current setwidth of the character.

The numerous small, filled boxes in the edit window are the control points for the shapes.

The column of icons at the left side of the window are the operator icons. You select one of these icons to invoke an editing function. (The rest of this chapter explains the action of these icons.)

All of the shape operator icons will cause part or all of the character window to redraw. Bezier curves are rendered in the edit window using line segment approximation. The BezierResolution command in the Options menu controls how fine the line segments should be. Low resolution is quick and adequate for highly interactive editing. If you want to view the curves in high resolution, which will be slow in rendering, choose the high option.

**View Preference**

To toggle the information displayed in the edit window, you can click on View Preferences in the Option menu. In the View Preferences dialog:

- **Top/Bottom** Draws the horizontal lines representing the maximum ascender and minimum descender for the entire font.
- **Ascender/Descender** Draws the horizontal lines representing the nominal ascender and descender of the font. These two numbers are specified in the font header.
- **x-Height** Draws the horizontal line representing the x-Height of the font. The x-Height is specified in the font header.
- **H-height** Draws the horizontal line representing the H-height of the font. The H-height is specified in the font header.
Bezier polygon  Draws the line segments connecting the control points of the Bezier curves of the character.

Figure 18-3
View Preference Dialog Box

Operator Icons

The next few sections describe the operator icons that appear to the left of the edit window. Several of these operator icons perform several actions. You can specify additional actions by holding down keys on the keyboard while clicking or releasing.

Some operations display the bounding box of the shape being manipulated and wait for you to confirm the operation. You confirm the operation by clicking on the left mouse button; you can cancel the operation by clicking on the right mouse button.

Moving Control Points

To move a control point, click on the Move Control Points operator, then point the cursor to the inside of the rectangular handle that encloses the control point. Press the left mouse button and drag the control point to where you want to put it. Where you release the button will be the new position of the control point.

When working with a low- or medium-resolution screen, it is not always possible to move a control point to an exact coordinate point, such as (100, 100). You can examine or alter the coordinate of a control point to the exact values you desire by holding down the Ctrl key and clicking on the control point handle whose coordinates you wish to change or examine. A dialog box will pop up for data entry.

You can undo most of the operations performed by the operators by choosing Undo in the Outline menu.
If you want a curve to flow smoothly at a segment juncture (between two Bezier curves, or between a curve and a line), hold the key and click on the handle of the control point that joins the two segments. FEDIT will adjust the coordinates of that point to ensure a first-order continuity between the two segments. You should apply this operator only when the adjoining segments are sufficiently close to continuity. If the curve turns sharply at the control point in question, using this operator will most likely yield undesirable results.

**Deleting a Segment**

To delete a segment in a shape, click on the Delete Segment operator, then click on the control point that ends the segment you want to delete. You cannot delete a Bezier control point that is not the end point of a curve.

To merge two Bezier curves into one, hold down the key, then click on the handle of the control point connecting the two curves. Not all pairs of curves can be merged successfully while retaining the original shapes of the two curves. If the resulting curve has a significantly different shape than the original ones, undo the change (Undo is in the OutLine menu).

**Shape Mutation**

To change a line to a curve or to change a curve to a line, click on the Shape Mutation operator.

Click near the segment you wish to mutate. FEDIT shows you the bounding rectangle of that segment. Click the left button again to confirm the change or click the right button to cancel the operation.

If the segment is a line, FEDIT replaces it with a Bezier curve that has two control points along the line. If the segment is a curve, FEDIT deletes its control points; the curve's end points become the end point for the line.

**Mitosis**

The mitosis operator breaks a segment into two segments of the same kind. Click near the segment you wish to duplicate, then click the left button anywhere in the segment where you want it broken, or click the right button to cancel the operation.
Adding a Rectangular Shape

The Rectangle operator allows you to create a rectangle in the edit window. If you hold the left button while dragging, you change the size of the rectangle; if you hold down both the left and right buttons, you can move the entire rectangle around. The rectangle consists of four line segments running in clockwise direction.

Adding an Oval Shape

The Oval operator allows you to create an oval in the edit window. While you click and drag the mouse, FEDIT draws a rectangle. When you release, FEDIT creates an oval that consists of four Bezier curve segments running in the clockwise direction.

The oval is bounded by the rectangle you saw while dragging.

Shape Transformation

The Shape Transformation operator is the most complex and versatile of the group. You use this operator to duplicate, move, scale, rotate, or shear a group of shapes.

To use the Shape Transformation operator:

1. Drag out a rectangle that encloses all the shapes you want to alter. The bounding rectangle that encloses all the selected shape will be highlighted. As usual, you can cancel the operation at this point by clicking the right button. Note that in order for a shape to be selected, all the control points in that shape must fall into the rectangle.

2. At this point, you choose whether you want to perform operations on the selected shapes or on a duplicate of the selected shapes. If you hold down the \texttt{Shift} key, FEDIT makes a copy of the selected shapes before performing subsequent transformations. If you do not hold down the \texttt{Shift} key, all transformations will be applied to the selected shapes.

3. To move the selected shapes, click the left mouse button and move the mouse; the shapes will move with the mouse. Release the button when you have positioned the shapes where you want them. As described above, you can copy the shapes by holding the \texttt{Shift} key while clicking and dragging.

4. To scale the selected shapes, hold the \texttt{Ctrl} key down while dragging out a scaling rectangle. The selected shapes will be scaled to fit into the rectangle you dragged out.

5. To rotate the selected shapes, hold the \texttt{Alt} key and the left mouse button while dragging with the mouse. You move the mouse in a motion reminiscent of traversing a circle around the center of the rectangle bounding the selected shapes.
To shear the selected shapes in the x-direction, hold the left arrow or right arrow key while you hold the left button and drag.

To shear the selected shapes in the y-direction, hold the up arrow or down arrow key while you hold the left button and drag.

While dragging out a rectangle or controlling the angle of rotation or shear, the intermediate transformed bounding rectangle (in the case of rotation or shearing, a bounding polygon) can be moved around by holding both the left and right mouse buttons down.

Deleting a Shape

To delete a shape, click on the Delete operator, then drag out a rectangle that encloses the shapes you want to delete. The shapes about to be deleted will be highlighted. Click the left button to confirm deletion or the right button to cancel the operation. Again, all control points of a shape must fall into the enclosing rectangle to be selected for deletion.

Changing the Setwidth

To change the setwidth for a character, click on the SetWidth operator. When you click on the SetWidth operator, FEDIT rescales the edit window so that the entire character and the setwidth line are visible.

To move the setwidth line, click inside the setwidth line's control handle (at the base line) and drag it along the x-axis.

You can also change the setwidth directly by entering the number in the Info dialog in the Character menu.

Viewing and Altering the Winding Direction

To view or alter the winding direction, click on the Winding Direction operator.

When you select this icon, FEDIT redraws the edit window to show the winding directions of the shapes (with arrows running along the shapes instead of the usual control point handles).

To reverse the winding direction in a shape, click near the shape. FEDIT highlights the selected shape. Click the left button again to confirm the change or the right button to cancel the selection.

In general, if one shape is totally enclosed within another, you should make sure that the outer shape winds in the counter-clockwise direction while the inner one winds in the clockwise direction, as shown in the figure on the right. The winding direction is very important in the shape merging operator (described next).
**Merging Shapes**

You use the Merge operator to merge two or more intersecting shapes. To merge shapes, drag out a rectangle that encloses all the shapes you want to merge. FEDIT highlights the bounding rectangle of the selected shapes. Click the left button to confirm or the right button to cancel the operation.

If you create two or more intersecting shapes, you will notice that the odd-even rule of filling causes the areas where the shapes intersect to be filled with white, rather than black. This is usually not desirable.

The Merge operator allows you to merge intersecting shapes so the resulting shapes have the same outline, but no longer intersect.

In Figure 18-4, the original shapes (A) consist of two pairs of intersecting concentric circles (four shapes). When these shapes are merged, FEDIT creates six shapes (B): the shape labeled 1 traces the arcs of the two outer-most circles; the other five shapes trace parts of the inner circles.

![Figure 18-4](image)

Shape merging is useful in eliminating intersecting regions where pixels are not rendered by the even-odd rule of filling. In Figure 18-5 above, the interlocked concentric circles will yield (C). By merging the shapes and then deleting the shapes labelled 3 and 5 in (B), we obtain (D).

To illustrate the importance of winding directions in shape merging, suppose we alter the winding directions of the original concentric circles such that all four circles wind in the clockwise direction, shown in Figure 18-6. When we merge these shapes, FEDIT produces only four merged shapes. The outer one is the same as that of the previous merging, but the inner ones are quite different.
Editing Hints

When characters are rendered at low resolution, they often suffer from pixel dropout or disproportionate weights on strokes. A 10-point, 72 dpi screen font is considered a low resolution font. A hint is data structure that helps low resolution rendering to minimize these problems.

Each character can have numerous hints associated with it. The proprietary hinting technology used in FEDIT is licensed from Digital Typeface Corporation. While we cannot document the hinting technology in full detail, we can explain enough to allow you to use the hint editing features of FEDIT.

- A hint controls the line widths of horizontal and vertical lines, or optionally, the line widths of curves where the tangent is horizontal or vertical. An x-hint controls x-coordinates (that is, the width of vertical lines). A y-hint controls y-coordinates (horizontal line width). Thus a character can have as many hints as there are horizontal and vertical line segments pairs (remember that a rendered “line” is actually bounded by two segments).

- A hint data structure has position, range, and length components. A hint affects all shapes that fall within the band defined by its range. The length component specifies the width of the shape when the font is rendered.

The Hint Editing Window

You bring up the hint editing window by choosing the Hint command under the OutLine menu. Figure 18-7 shows the hint editing window.

In the hint editing window, the shape operator icons are replaced by the hint operator icons. The control points or arrows on the character are not displayed in the hint editing window.
The x-hints with decreasing coordinates appear along the top edge of the edit window; x-hints with increasing coordinates appear along the bottom edge. The y-hints with decreasing coordinates appear along the left edge; y-hints with increasing coordinates appear on the right edge. See the explanation of hint coordinates below.

#### Altering a Hint

Each hint has three control handles that represent the starting and ending coordinate values of the hint (solid rectangles) and the width control (hollow rectangle).

To alter a hint, click on the Alter Hint operator, then point the cursor at one of the control handles and drag.

Hints are effective only when rendering at low resolution. To view the effects of the hints, you should scale the sample windows to about 10 or 12 points. In addition, turn on the AutoRedraw option under the Option menu. The sample windows will continuously redraw as you modify the hints. Turn on the discarded option only when the sample windows are small; larger-sized characters take longer to render and affect the response of the mouse in altering hints.
Creating a Hint

To create a hint, click on either the x- or y-hint operator. Then click in the edit window to create the hint. If you’re creating an x-hint, the x-coordinate of the mouse at the time you click on the left button will be starting position of the hint. If you are creating a y-hint, the y-coordinate of the mouse will be the starting position.

Deleting a Hint

To delete a hint, click on the Delete hint operator, then click on the hint you want to delete. FEDIT highlights the selected hint. Click the left button again to confirm the deletion or the right button to cancel the operation.

Editing Bitmaps

To obtain ultimate fidelity of a font at low resolution, a hand-tuned bitmap is unavoidable. In the PenPoint operating system, a bitmap (or bitmaps) is considered an integral part of a font, tied closely to the rest of the font’s data structures. It is not an entity that can exist independently (that is, there is no provision for a bitmap font without outline data). Additionally, PenPoint has no mechanism by which you can attach an externally created bitmap to an outline font. You must use FEDIT to create and render a bitmap initially, then edit it to your heart’s content.

Because there is a close binding between the outline metrics (in font units) and bitmap pixel metrics (in pixel units), creating and editing bitmaps should be considered finishing touches in the font creation process.

The worst thing you can do is to alter or create a character so that its vertical metrics exceed the current maximum ascender or minimum descender of a font for which you have already built bitmaps. FEDIT is currently not very flexible in regard to support this kind of activity. However, FEDIT does provide character cell alteration tools (described below) to handle this scenario. You won’t have to start building the bitmaps from scratch, but it is still a tedious process which should be avoided.

Creating a Bitmap

To create a bitmap, choose the Create command under the Bitmap menu. A dialog box pops up for entering the specification of the bitmap:

Aspect ratio  Enter the aspect ratio of the device for which the bitmap font is being created. Use small integers. For example, most nine-pin dot matrix printers have an aspect ratio of 2:1; enter x=2 and y=1.
**Pixel Height** Enter the number of scanlines per 1000 font units. For example, for a 12-point font on a 300 dpi printer, this number will be \((12 / 72) \times 300 = 50\) pixels.

**Rotation** Specify any one of the four rotations. If you are creating bitmaps with 1:1 aspect ratio, you need only to create the bitmap with zero rotation, even if you are planning to use your font in more than one rotation. The PenPoint font subsystem will perform rotation for you automatically. Angles are measured counter-clockwise from the x-axis.

To maintain consistency between bitmaps and outline data, you should note the following:

- Altering the shapes of a character does not affect any existing bitmaps. To update the existing bitmaps to the character’s new appearance, select that character in pixel editing mode (described below) and execute the Re-render Char command under the Bitmap menu. Repeat this procedure for each bitmap you have created.

- If you altered the height of a character to the extent that the character no longer fits the height of the bitmap (the height of a bitmap is the y-dimension for 0 or 180 rotation, x-dimension for 90 or 270), you would have to manually add rows (or columns) using the Edit Char command (described below).

- If you delete a character from the font, FEDIT automatically deletes the bitmap cell for that character from each existing bitmap.

- If you add a character to the font, FEDIT appends a bitmap cell for that character to the end of each bitmap. The width of each cell is computed based on the initial setwidth you assigned to the character. After you have added shapes to the new character, edit that character for each bitmap in pixel editing mode and execute the Re-render Char command. Because a new character did not have any shape at the time it was created, no pixel was drawn to the new character cells and it will be invisible. Click on the blank space just past the end of the character list to select the new character. The bitmaps will be sorted in ascending order of AFII numbers when they are saved. Thus, after you save and re-open the font file, the new character will appear in a new place.

**Deleting a Bitmap**

To delete a bitmap, choose Edit from the Bitmap menu. Select a bitmap from the list box. Click on the Delete button. Once you delete a bitmap, it cannot be undone; think twice before you do it.
Pixel Editing

To edit a bitmap, choose Edit from the Bitmap menu. Select the bitmap you want to edit and click on the Edit button. The bitmap edit window will appear. The list of characters in the bitmap is shown on the bottom of the main window. You can scroll the character list with the horizontal scroll bar. If the bitmap you are editing is rotated 90 or 270, the character list appears on the right edge of the main window and you scroll it with the vertical scroll bar instead.

Select the character you want to edit by clicking on that character in the list. Click on a pixel cell in the pixel grid of the character to invert its state. FEDIT updates the character image in the character list instantly so you can monitor its appearance as you modify the pixels.

The initial state of the bitmap is obtained by rendering each character in the font from the outline data.
Altering Cell Dimensions

To alter the dimension of a character cell, choose Edit Cell from the Bitmap menu. You may delete or insert a border row or column.

A bitmap with a 0° or 180° rotation is said to be row major (that is, every character in the bitmap has the same number of rows). If you add or delete a row, the entire bitmap will be affected. A warning message will appear to ask for confirmation. Changing the number of columns will affect only the character you are editing.

A bitmap with a 90° or 270° rotation is said to be column major. Changing the number of columns will affect the entire bitmap while changing the number of rows will affect only the character you are editing.

In general, you should not alter the dimensions of a character by more than one pixel (if any at all). Under some circumstances, the font subsystem in PenPoint computes a character's pixel metrics by scaling its metrics in font units. Fidelity problems can occur if the scaled metrics differ too greatly from the actual metrics in the bitmap.

Miscellaneous Functions

Copying to the Clipboard

You can transfer shape and hint data (but not bitmap data) using the Windows clipboard. You must be in the shape editing mode to enable copying.

Choose Copy All from the OutLine menu. All the shape and hint data will be copied to the clipboard.

You may also copy a portion of the character by using one of three methods:
- Choose Selective Copy from the OutLine menu.
- Drag out a rectangle in the shape editing window. All the shapes and hint data enclosed in this region are now selected for copying. The bounding rectangle for the selected shapes and hints will be highlighted.
- Click the left button to confirm the operation; click the right button to cancel it.

Pasting from the Clipboard

Pasting is enabled if there are shape or hint data in the clipboard; you must be in the shape editing or hint editing mode.

Choose Paste Absolute if you want to copy the clipboard data to the current character in the same position from which the data were copied.

Choose Paste Relative if you want to copy the clipboard data to a new position in the current character. In this mode, after you have selected the operation from the menu, press the left mouse button in the editing window. The bounding rectangle
for the clipboard data will appear. Move this rectangle to the location where you want to place the shapes or hints. Release the button.

You can undo a paste operation.

**Subset Saving**

You can save a subset of the characters and bitmaps in a font to a different file. Choose Save Subset from the File menu. FEDIT prompts for the output file name as well as a file containing the list of characters and bitmaps you wish to save. The format of this file is a list of AFII numbers and bitmap specifications. AFII numbers are simply signed integers. Bitmap specification starts with a colon (:) followed by the width, height, and rotation of the bitmap. For example, the following list saves the characters A to E and the bitmap with 12 x 12, 0 rotation:

```
501 502 503 504 505 :12 12 0
```

The order of the entries in the list is unimportant and new lines are permitted in the file. The saved bitmaps will contain only characters you have chosen in the list.

**Examining and Editing the Font Header**

To examine and edit the global information about the font, choose the Font Header or the Font Attribute command under the File menu. You can modify any fields that have a border; fields without a border are maintained by FEDIT.

Table 18-1 describes the fields in the Font Header window; Table 18-2 describes the fields in the Font Attribute window. All metrics are in font units.

---

**Table 18-1: Font Header Window**

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Notice</td>
<td>URW Roman is a trademark of URW</td>
</tr>
<tr>
<td>Font name</td>
<td>URW Roman</td>
</tr>
<tr>
<td>Character count</td>
<td>95</td>
</tr>
<tr>
<td>Pitch</td>
<td>0</td>
</tr>
<tr>
<td>Under position</td>
<td>-105</td>
</tr>
<tr>
<td>Ascender</td>
<td>695</td>
</tr>
<tr>
<td>Max ascender</td>
<td>738</td>
</tr>
<tr>
<td>Max right</td>
<td>929</td>
</tr>
<tr>
<td>x-Height</td>
<td>444</td>
</tr>
<tr>
<td>OK</td>
<td>Cancel</td>
</tr>
</tbody>
</table>

---

**Figure 18-9: Font Header Window**
<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Notice</td>
<td>Copyright notice.</td>
</tr>
<tr>
<td>Font name</td>
<td>Name of the font.</td>
</tr>
<tr>
<td>Character count</td>
<td>Number of characters in this font.</td>
</tr>
<tr>
<td>Version</td>
<td>Version number of the font.</td>
</tr>
<tr>
<td>Fixed pitch</td>
<td>0 if the font is proportionally spaced; character width if the font is monospaced.</td>
</tr>
<tr>
<td>Space width</td>
<td>Width of the space character.</td>
</tr>
<tr>
<td>Under position</td>
<td>Offset from the base line where an underline will be placed. This is a negative number.</td>
</tr>
<tr>
<td>Under thickness</td>
<td>Width of an underline.</td>
</tr>
<tr>
<td>Ascender</td>
<td>A positive number giving the y-offset of the ascender line. Typically, the top of a capital letter. This value is the opinion of the font designer, and does not necessarily correspond to the tallest character in the font.</td>
</tr>
<tr>
<td>Descender</td>
<td>A negative number giving the y-offset of the descender line. Typically, the bottom of a j. This value is the opinion of the font designer, and does not necessarily correspond to the deepest descender in the font.</td>
</tr>
<tr>
<td>x-height</td>
<td>A positive number giving the y-offset of the x-height line. Typically, the top of a x. This value is the opinion of the font designer, and does not necessarily correspond to the actual x.</td>
</tr>
<tr>
<td>h-height</td>
<td>A positive number giving the y-offset of the h-height line. Typically, the top of an h. This value is the opinion of the font designer, and does not necessarily correspond to the actual h.</td>
</tr>
<tr>
<td>Max ascender, Min descender, Max right, Min left</td>
<td>The smallest and largest x and y values found in the font. These can be used to construct a rectangle that would enclose any character from the font. These values are not the opinion of the font designer; they are computed automatically from the character definitions and cannot otherwise be edited.</td>
</tr>
<tr>
<td>Bitmaps</td>
<td>The number of bitmaps in the font.</td>
</tr>
</tbody>
</table>

**Table 18-1**

**Font Header Window Fields**

**Figure 18-10**

**Font Attribute Window**
Table 18-2
Font Attribute Window Fields

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Font ID</td>
<td>A 118-bit number assigned by GO.</td>
</tr>
<tr>
<td>Group</td>
<td>The group characterizes the font design; this is used by software to find a “best match” when requested font is unavailable.</td>
</tr>
<tr>
<td>Weight</td>
<td>Specifies the stroke weight of the font.</td>
</tr>
<tr>
<td>Aspect</td>
<td>Specifies the designed aspect ratio (not the device aspect ratio) of the font.</td>
</tr>
<tr>
<td>Monospaced</td>
<td>Check this box if the font is designed to be a monospaced font.</td>
</tr>
<tr>
<td>Italic</td>
<td>Check this box if the font is an italic font.</td>
</tr>
</tbody>
</table>

**Examining Text Samples**

To view a sample of ASCII text rendered from the font, choose Show Sample from the Character menu. FEDIT displays a text sample window. Choose the same command again to take down the window.

**Using Fonts in Documentation**

If you need use your FEDIT font in printed documentation, you should use FEDIT to save your font as an Adobe Type I font (described below).

If you are printing to a PostScript printer, you can download the font to the printer or, depending on your word processor, you can configure the font in the word processor and enable it to download the font.
If you are printing to a PCL printer, you need to use a word processor or page-layout program that uses Adobe Type Manager under either Microsoft Windows or Macintosh. When you use Adobe Type Manager, your font will be rendered on screen. When you print to a PCL printer, Adobe Type Manager creates PCL bitmaps for your font and downloads them to the printer.

### Adobe Type I Fonts

An Adobe Type I font is a Postscript program; as such, a robust reading of such a font requires an implementation of a Postscript interpreter. FEDIT does not implement such an interpreter. Rather, it employs heuristics based on common conventions described in the book *Adobe Type I Font Format* published by Adobe Systems, Inc. One can construct a perfectly legitimate Adobe Type I font file without regard to these conventions. GO does not guarantee that FEDIT will load all Adobe Type I font files successfully. In addition, FEDIT currently imposes a size limit of 64K on such a font file.

Certain incompatibilities exist between the hinting technologies of Adobe Systems Inc. and Digital Typeface Corporation. Where an Adobe hint feature does not apply to PenPoint, it is discarded. Thus, there may a loss of information in loading an Adobe Type I font.

Additionally, each character in an Adobe font has a name either defined implicitly by standard encoding, or explicitly by a custom encoding. FEDIT, on the other hand, identifies a character by its AFII number. When FEDIT loads an Adobe font, it maps an Adobe character by its character code (0-255) to an AFII number according to Table 18-3, below. The character names are discarded.

### Saving an Adobe Font

Additional information is necessary to create an Adobe Type I font. The dialog box show here pops up when you attempt to save the currently opened font as an Adobe font. Table 18-3 describes the fields in the dialog box. The description assumes a certain familiarity with the PostScript language.
Adobe Font Fields

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Notice</td>
<td>Copyright notice of the font. The /Notice entry in the created /FontInfo dictionary is set to this string.</td>
</tr>
<tr>
<td>Full name</td>
<td>Full name of the font. The /FullName entry in the created /FontInfo dictionary is set to this string.</td>
</tr>
<tr>
<td>Font name</td>
<td>Name of the font. The /FontName entry in the created font dictionary is set to this string. This is the name you would use with the findfont operator in your PostScript program.</td>
</tr>
<tr>
<td>Family</td>
<td>The /FamilyName entry in the created /FontInfo dictionary is set to this string.</td>
</tr>
<tr>
<td>Unique ID</td>
<td>The /UniqueID entry in the created font dictionary is set to this string.</td>
</tr>
<tr>
<td>Make AFM</td>
<td>Instructs FEDIT to output an Adobe font metrics file also. The extension of the metric file is always set to .AFM and .PFB for the font body.</td>
</tr>
<tr>
<td>Encoding</td>
<td>Choose the encoding of the font. If you choose the Adobe standard encoding, FEDIT checks that the AFII number of each character in the font has a mapping in the Adobe encoding. If any character fails the test, an error message will appear and the file will not be saved. In addition, the Adobe standard encoding can have a maximum of 256 characters only. The custom sequential encoding is primarily for a font with graphics symbols that don’t have standard AFII to Adobe mappings. In this mode, each character in the font, in ascending AFII numbers, is assigned to the standard 96-character ASCII encoding, starting with the space character. Thus, you cannot encode a font with more than 96 characters in this mode.</td>
</tr>
</tbody>
</table>

Adobe Standard Encoding to AFII Mappings

Table 18-4 shows the mapping between an Adobe character and its assumed AFII code in the Adobe standard encoding mode. The column labeled “encoded?” simply indicates whether a character is designated to be encoded or not according to the Postscript specification. FEDIT does not pay attention to this property.

<table>
<thead>
<tr>
<th>Adobe Character Code</th>
<th>Adobe Character Name</th>
<th>AFII Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>space</td>
<td>32</td>
</tr>
<tr>
<td>33</td>
<td>exclaim</td>
<td>33</td>
</tr>
<tr>
<td>34</td>
<td>quotedbl</td>
<td>34</td>
</tr>
<tr>
<td>35</td>
<td>numbersign</td>
<td>35</td>
</tr>
<tr>
<td>36</td>
<td>dollar</td>
<td>164</td>
</tr>
<tr>
<td>37</td>
<td>percent</td>
<td>37</td>
</tr>
<tr>
<td>38</td>
<td>ampersand</td>
<td>38</td>
</tr>
<tr>
<td>39</td>
<td>quoteright</td>
<td>39</td>
</tr>
<tr>
<td>40</td>
<td>parenleft</td>
<td>40</td>
</tr>
<tr>
<td>41</td>
<td>parenright</td>
<td>41</td>
</tr>
<tr>
<td>42</td>
<td>asterisk</td>
<td>42</td>
</tr>
<tr>
<td>43</td>
<td>plus</td>
<td>43</td>
</tr>
</tbody>
</table>

continued
<table>
<thead>
<tr>
<th>Adobe Character Code</th>
<th>Adobe Character Name</th>
<th>AFI Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>44</td>
<td>comma</td>
<td>44</td>
</tr>
<tr>
<td>45</td>
<td>hyphen</td>
<td>45</td>
</tr>
<tr>
<td>46</td>
<td>period</td>
<td>46</td>
</tr>
<tr>
<td>47</td>
<td>slash</td>
<td>47</td>
</tr>
<tr>
<td>48</td>
<td>zero</td>
<td>48</td>
</tr>
<tr>
<td>49</td>
<td>one</td>
<td>49</td>
</tr>
<tr>
<td>50</td>
<td>two</td>
<td>50</td>
</tr>
<tr>
<td>51</td>
<td>three</td>
<td>51</td>
</tr>
<tr>
<td>52</td>
<td>four</td>
<td>52</td>
</tr>
<tr>
<td>53</td>
<td>five</td>
<td>53</td>
</tr>
<tr>
<td>54</td>
<td>six</td>
<td>54</td>
</tr>
<tr>
<td>55</td>
<td>seven</td>
<td>55</td>
</tr>
<tr>
<td>56</td>
<td>eight</td>
<td>56</td>
</tr>
<tr>
<td>57</td>
<td>nine</td>
<td>57</td>
</tr>
<tr>
<td>58</td>
<td>colon</td>
<td>58</td>
</tr>
<tr>
<td>59</td>
<td>semicolon</td>
<td>59</td>
</tr>
<tr>
<td>60</td>
<td>less</td>
<td>60</td>
</tr>
<tr>
<td>61</td>
<td>equal</td>
<td>61</td>
</tr>
<tr>
<td>62</td>
<td>greater</td>
<td>62</td>
</tr>
<tr>
<td>63</td>
<td>question</td>
<td>63</td>
</tr>
<tr>
<td>64</td>
<td>at</td>
<td>64</td>
</tr>
<tr>
<td>65</td>
<td>A</td>
<td>65</td>
</tr>
<tr>
<td>66</td>
<td>B</td>
<td>66</td>
</tr>
<tr>
<td>67</td>
<td>C</td>
<td>67</td>
</tr>
<tr>
<td>68</td>
<td>D</td>
<td>68</td>
</tr>
<tr>
<td>69</td>
<td>E</td>
<td>69</td>
</tr>
<tr>
<td>70</td>
<td>F</td>
<td>70</td>
</tr>
<tr>
<td>71</td>
<td>G</td>
<td>71</td>
</tr>
<tr>
<td>72</td>
<td>H</td>
<td>72</td>
</tr>
<tr>
<td>73</td>
<td>I</td>
<td>73</td>
</tr>
<tr>
<td>74</td>
<td>J</td>
<td>74</td>
</tr>
<tr>
<td>75</td>
<td>K</td>
<td>75</td>
</tr>
<tr>
<td>76</td>
<td>L</td>
<td>76</td>
</tr>
<tr>
<td>77</td>
<td>M</td>
<td>77</td>
</tr>
<tr>
<td>78</td>
<td>N</td>
<td>78</td>
</tr>
<tr>
<td>79</td>
<td>O</td>
<td>79</td>
</tr>
<tr>
<td>80</td>
<td>P</td>
<td>80</td>
</tr>
<tr>
<td>81</td>
<td>Q</td>
<td>81</td>
</tr>
</tbody>
</table>

continued
<table>
<thead>
<tr>
<th>Adobe Character Code</th>
<th>Adobe Character Name</th>
<th>AFI Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>82</td>
<td>R</td>
<td>82</td>
</tr>
<tr>
<td>83</td>
<td>S</td>
<td>83</td>
</tr>
<tr>
<td>84</td>
<td>T</td>
<td>84</td>
</tr>
<tr>
<td>85</td>
<td>U</td>
<td>85</td>
</tr>
<tr>
<td>86</td>
<td>V</td>
<td>86</td>
</tr>
<tr>
<td>87</td>
<td>W</td>
<td>87</td>
</tr>
<tr>
<td>88</td>
<td>X</td>
<td>88</td>
</tr>
<tr>
<td>89</td>
<td>Y</td>
<td>89</td>
</tr>
<tr>
<td>90</td>
<td>Z</td>
<td>90</td>
</tr>
<tr>
<td>91</td>
<td>bracketleft</td>
<td>91</td>
</tr>
<tr>
<td>92</td>
<td>backslash</td>
<td>92</td>
</tr>
<tr>
<td>93</td>
<td>bracketright</td>
<td>93</td>
</tr>
<tr>
<td>94</td>
<td>asciicircum</td>
<td>94</td>
</tr>
<tr>
<td>95</td>
<td>underscore</td>
<td>95</td>
</tr>
<tr>
<td>96</td>
<td>quoteleft</td>
<td>96</td>
</tr>
<tr>
<td>97</td>
<td>a</td>
<td>97</td>
</tr>
<tr>
<td>98</td>
<td>b</td>
<td>98</td>
</tr>
<tr>
<td>99</td>
<td>c</td>
<td>99</td>
</tr>
<tr>
<td>100</td>
<td>d</td>
<td>100</td>
</tr>
<tr>
<td>101</td>
<td>e</td>
<td>101</td>
</tr>
<tr>
<td>102</td>
<td>f</td>
<td>102</td>
</tr>
<tr>
<td>103</td>
<td>g</td>
<td>103</td>
</tr>
<tr>
<td>104</td>
<td>h</td>
<td>104</td>
</tr>
<tr>
<td>105</td>
<td>i</td>
<td>105</td>
</tr>
<tr>
<td>106</td>
<td>j</td>
<td>106</td>
</tr>
<tr>
<td>107</td>
<td>k</td>
<td>107</td>
</tr>
<tr>
<td>108</td>
<td>l</td>
<td>108</td>
</tr>
<tr>
<td>109</td>
<td>m</td>
<td>109</td>
</tr>
<tr>
<td>110</td>
<td>n</td>
<td>110</td>
</tr>
<tr>
<td>111</td>
<td>o</td>
<td>111</td>
</tr>
<tr>
<td>112</td>
<td>p</td>
<td>112</td>
</tr>
<tr>
<td>113</td>
<td>q</td>
<td>113</td>
</tr>
<tr>
<td>114</td>
<td>r</td>
<td>114</td>
</tr>
<tr>
<td>115</td>
<td>s</td>
<td>115</td>
</tr>
<tr>
<td>116</td>
<td>t</td>
<td>116</td>
</tr>
<tr>
<td>117</td>
<td>u</td>
<td>117</td>
</tr>
<tr>
<td>118</td>
<td>v</td>
<td>118</td>
</tr>
<tr>
<td>119</td>
<td>w</td>
<td>119</td>
</tr>
<tr>
<td>Adobe Character Code</td>
<td>Adobe Character Name</td>
<td>AFI Code</td>
</tr>
<tr>
<td>----------------------</td>
<td>----------------------</td>
<td>----------</td>
</tr>
<tr>
<td>120</td>
<td>x</td>
<td>120</td>
</tr>
<tr>
<td>121</td>
<td>y</td>
<td>121</td>
</tr>
<tr>
<td>122</td>
<td>z</td>
<td>122</td>
</tr>
<tr>
<td>123</td>
<td>bracelleft</td>
<td>123</td>
</tr>
<tr>
<td>124</td>
<td>bar</td>
<td>124</td>
</tr>
<tr>
<td>125</td>
<td>braceright</td>
<td>125</td>
</tr>
<tr>
<td>126</td>
<td>asciitilde</td>
<td>126</td>
</tr>
<tr>
<td>161</td>
<td>exclamdown</td>
<td>161</td>
</tr>
<tr>
<td>162</td>
<td>cent</td>
<td>162</td>
</tr>
<tr>
<td>163</td>
<td>sterling</td>
<td>163</td>
</tr>
<tr>
<td>164</td>
<td>fraction</td>
<td>164</td>
</tr>
<tr>
<td>165</td>
<td>yen</td>
<td>165</td>
</tr>
<tr>
<td>166</td>
<td>florin</td>
<td>166</td>
</tr>
<tr>
<td>167</td>
<td>section</td>
<td>167</td>
</tr>
<tr>
<td>168</td>
<td>currency</td>
<td>168</td>
</tr>
<tr>
<td>169</td>
<td>quotesingle</td>
<td>169</td>
</tr>
<tr>
<td>170</td>
<td>quotedblleft</td>
<td>170</td>
</tr>
<tr>
<td>171</td>
<td>guillemotleft</td>
<td>171</td>
</tr>
<tr>
<td>172</td>
<td>guilsingleleft</td>
<td>172</td>
</tr>
<tr>
<td>173</td>
<td>guilsinglright</td>
<td>173</td>
</tr>
<tr>
<td>174</td>
<td>fi</td>
<td>174</td>
</tr>
<tr>
<td>175</td>
<td>fl</td>
<td>175</td>
</tr>
<tr>
<td>177</td>
<td>endash</td>
<td>177</td>
</tr>
<tr>
<td>178</td>
<td>dagger</td>
<td>178</td>
</tr>
<tr>
<td>179</td>
<td>daggerdbl</td>
<td>179</td>
</tr>
<tr>
<td>180</td>
<td>periodcentered</td>
<td>180</td>
</tr>
<tr>
<td>182</td>
<td>paragraph</td>
<td>182</td>
</tr>
<tr>
<td>183</td>
<td>bullet</td>
<td>183</td>
</tr>
<tr>
<td>184</td>
<td>quotesinglebase</td>
<td>184</td>
</tr>
<tr>
<td>185</td>
<td>quotedblbase</td>
<td>185</td>
</tr>
<tr>
<td>186</td>
<td>quotedblright</td>
<td>186</td>
</tr>
<tr>
<td>187</td>
<td>guillemotright</td>
<td>187</td>
</tr>
<tr>
<td>188</td>
<td>ellipsis</td>
<td>188</td>
</tr>
<tr>
<td>189</td>
<td>perthousand</td>
<td>189</td>
</tr>
<tr>
<td>191</td>
<td>questiondown</td>
<td>191</td>
</tr>
<tr>
<td>193</td>
<td>grave</td>
<td>193</td>
</tr>
<tr>
<td>194</td>
<td>acute</td>
<td>194</td>
</tr>
<tr>
<td>195</td>
<td>circumflex</td>
<td>195</td>
</tr>
</tbody>
</table>
Adobe Character Code | Adobe Character Name | Afii Code
---|---|---
196 | tilde | 196
197 | macron | 197
198 | breve | 198
199 | dotaccent | 199
200 | dieresis | 200
202 | ring | 202
203 | cedilla | 203
205 | hungarumlaut | 205
206 | ogonek | 206
207 | caron | 207
208 | emdash | 208
225 | AE | 225
227 | ordfeminine | 227
232 | Lslash | 232
233 | Oslash | 233
234 | OE | 234
235 | ordmasculine | 235
241 | ae | 241
245 | dotlessi | 245
248 | lslash | 248
249 | oslash | 249
250 | oe | 250
251 | germandbls | 251

Font File Formats

This section describes the two of the three font file formats supported by FEDIT. For information on the Adobe Type I format, see *Adobe Type I Font Format* published in 1990 by Adobe Systems, Inc.

The font file formats described here are:

- Nimbus-Q format
- PenPoint Packed Format

The Nimbus-Q Format

The Nimbus-Q font file has four main sections illustrated below:
Font Header

The Nimbus-Q font header has the following structure:

```c
typedef struct NIMBUSQ_HDR {
    long    signature;
    char    notice[80];
    char    fullName[80];
    char    fontName[80];
    char    family[80];
    char    weight[80];
    char    version[80];
    char    charSet[80];
    double  italicAngle;
    short   fixPitch;
    short   spaceWidth;
    short   underPosition;
    short   underThickness;
    short   hHeight;
    short   xHeight;
    short   ascender;
    short   descender;
    short   nChars;
} NIMBUSQ_HDR;
```

The font file signature is not under the control of GO. The currently known value is 131431010

AFII Number Array

The AFII number array holds the character IDs in the font. Each number is a 16-bit integer. The size of the table is specified by the nChars field in the font header.

Character Data File Positions

The character data file positions area is table of 32-bit file positions pointing to the character data. There are as many entries in the table as there are characters in the font.

Character Data

The start of each character data block is located by the file position table above. Each character data block consists of the following fields:

- Setwidth of the character (16-bit word).
- The x- and y-coordinates of the lower-left-hand corner of the character’s bounding rectangle (two 16-bit words).
- The x- and y-coordinates of the upper-right-hand corner of the character’s bounding rectangle (two 16-bit words).
- Number of x-hints in the character (16-bit word).
- As many x-hint structures as the number in the preceding field. Each hint contains a starting coordinate, an ending coordinate, and a length (3 16-bit words).
Number of y-hints in the character (16-bit word).
• As many y-hint structures as the number in the preceding field.
• Next comes the segment data. Each segment starts with a 16-bit code, followed by 0, 1, or 3 pairs of coordinate values, depending on the shape code:
  0 Move to the coordinate point that follows.
  1 Draw a line to the coordinate point that follows.
  2 Draw a curve to the 3 coordinate points that follow.
  3 End of shape—no data follows this code.

The PenPoint Packed Format

The GO packed format is used by the PenPoint font subsystem. It is a highly compressed format suitable for the PenPoint environment but not for programmers' mental health. The following list shows the file's main layout.

Font Header
URW Number Array
Bitmap Directory
Character Directory
Short Bezier Dictionary Pointer
Long Bezier Dictionary Pointer
Shape Data Pointer
Hint Data
Short Bezier Dictionary
Long Bezier Dictionary
Shape Data
Bitmap Data

Font Header
The GO font header has the following structure.

```c
typedef struct FONT_HDR {
    long signature;
    char notice[80];
    char fullName[80];
    short fontId;
    short attr;
    short version;
    short fixPitch;
    short spaceWidth;
    short underPosition;
    short underThickness;
    short xHeight;
    short hHeight;
    short ascender;
    short descender;
    short nChars;
    short maxTriples;
} FONT_HDR;
```
short nShapeRuns;
short bigModel;
short maxAscender;
short minDescender;
short maxRight;
short minLeft;
short reserved[14];
short nBitmaps;
}

Most of the fields in the font header have been described in a previous section. Here are descriptions of fields not documented before:

- The GO font file signature is 13143110.
- The font attribute word attrib has the following structure:

<table>
<thead>
<tr>
<th>group (4 bits)</th>
<th>weight (2 bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unassigned</td>
<td>Light</td>
</tr>
<tr>
<td>Venetian</td>
<td>Normal</td>
</tr>
<tr>
<td>Old style</td>
<td>Bold</td>
</tr>
<tr>
<td>Transitional</td>
<td>Extra bold</td>
</tr>
<tr>
<td>Modern roman</td>
<td>italic (1 bit)</td>
</tr>
<tr>
<td>Egyptian</td>
<td>Normal</td>
</tr>
<tr>
<td>San serif</td>
<td>Italic</td>
</tr>
<tr>
<td>Display roman</td>
<td>aspect (2 bits)</td>
</tr>
<tr>
<td>Script</td>
<td>Condensed</td>
</tr>
<tr>
<td>Graphic</td>
<td>Normal</td>
</tr>
<tr>
<td>Software defined</td>
<td>Extended</td>
</tr>
</tbody>
</table>

maxTriples contains the number of hints (x and y) of the character which has the maximum number of hints in the font.

- The fields bigModel and nShapeRuns are explained in the shape data section.

**AFII Number Array**

The AFII number array holds the character IDs in the font. Each number is a signed 16-bit integer. The size of the table is specified by the nChars field in the font header. The array is sorted in ascending order of the AFII numbers.

**Bitmap Directory**

The bitmap directory contains as many entries as there are bitmaps in the font. This section can be absent if there are no bitmaps in the font. Each entry describes a bitmap and contains the following fields:
Canonical width and height  The pair of values obtained by transforming (1000, 1000) by the matrix used for scaling the font when the bitmap was created (two 16-bit words).

Rotation  The angle of rotation of the bitmap (16-bit word). It can be any one of the values 0, 90, 180, or 360.

SizeX, SizeY  The actual width and height of the bitmap in pixel units (two 16-bit words).

Reserved  Reserved by GO for future use (16-bit word).

Ascender  The ascender of the bitmap in pixel units (16-bit word).

Row byte size  The number of bytes in a row of the bitmap, must be an even number (16-bit word).

Bitmap file position  The location in the file where the pixel map is stored (32-bit word).

Character Directory

The character directory contains as many character definition entry as there are characters in the file, plus one. The extra entry serves as a sentinel for the hint and shape indices calculations. Each entry has the following structure:

Setwidth  In font units (16-bit word).

Bounding rectangle  Two 24-bit packed number pairs for the lower-left and upper-right corners of the character’s bounding rectangle (6 bytes).

X-hint index  The index number of the starting x-hint data in the hint data section of the file (16-bit word). The number of x-hints for this character is obtained by subtracting this number from the next field (which can be 0).

Y-hint index  The index number of the starting y-hint data in the hint data section of the file (16-bit word). The number of y-hints for this character is obtained by subtracting this number from the x-hint index of the next character (which can be 0).

Shape index  The index number of the starting shape data in the shape data section (16-bit word). The number of shapes for this character is obtained by subtracting this number from the shape index of the next character.

There will be many occurrences of a 24-bit packed number pair in a font file. It consists of two signed 12-bit numbers, packed into 3 bytes and has the following structures:

byte 1  Low-order byte of n1.

byte 2  Low-order of n2.

byte 3  Hi-order nibble of n1, hi-order nibble of n2.

File Pointers

The next 3 fields are 32-bit file positions pointing to the short Bezier dictionary, the long Bezier dictionary, and the shape data section.
**Hint Data**

The hint data section contains all the hint data of the font. The hints are accessed by the hint data indices of the character definition. For example, if a character’s starting index for x-hints is n and it has 3 x-hints, then hint[n], hint[n+1], and hint[n+2] are the x-hints of the character.

Each hint data structure consists of a 24-bit packed number pair, encoding the starting and ending coordinates of the hint, and a 16-bit word for the length of the hint.

**Shape Data**

We will skip forward to describe the shape data section before the Bezier dictionaries. The shape data section contains two main subsections:

- The offset table.
- A series of variable-length shape blocks.

The offset table contains byte offsets to shape blocks. The offset is relative to the first byte past the end of the offset table. There are nShapeRuns (see font header) entries in the table. Each entry is a 16-bit word if the bigModel flag (see font header) is false, 32-bit word otherwise.

Each shape block contains:

- The starting point of a shape in 24-bit packed coordinate format.
- The shape data. All shape data are relative coordinate values, that is, begin with the starting point of the shape, the next point is computed by adding the values from the shape data to the current point as you decode the shape data. All shape data are signed values regardless of their bit length.
- An array of shape codes, each 4-bit long. The shape code dictates how the shape data are accessed and the array is arranged in a reverse order, that is, the first shape code is the last entry in the array. The shape codes are:

  0 Draw relative line with the 24-bit packed coordinate pair.
  1 Draw relative line with the 8 + 8 bit coordinate pair.
  2 Draw relative horizontal line with the 16-bit coordinate.
  3 Draw relative horizontal line with the 8-bit coordinate.
  4 Draw relative vertical line with the 16-bit coordinate.
  5 Draw relative vertical line with the 8-bit coordinate.
  6 Draw relative curve with the 3 24-bit packed coordinate pairs.
  7 Draw relative curve with the 3 8 + 8 bit coordinate pairs.
  8 Draw relative curve with the 16-bit index into the long Bezier dictionary.
  9 Draw relative curve with the 16-bit index into the short Bezier dictionary.
 10 Branch off to a shape block with the 16-bit index and draw n segments using data from that shape block. n is a 16-bit value that follows the shape index. Maintain the current point when branching off, do not switch to the starting point of the referenced shape block.
11 End of shape block. No data is associated with this code.
To decode a shape data block, use this algorithm:

- Keep two pointers p0 and p1. p0 points to the beginning of the shape block data (the byte past the starting point). p1 points to the end of the shape code array. Compute p1 by adding the difference of the byte offset of the next shape block and that of the current shape block, minus 1, to p0.

- Fetch a shape code with p1, and fetch the shape data with p0. The size of the shape data to fetch depends on the shape code, as listed in the table above. Recall a shape code is a 4-bit quantity. So the first shape code is the low-order 4 bits of the byte pointed to by p1.

- Process the data. Advance p0 with the size of the data just read. Decrement p1 by 4 bits. Repeat the process until the end-of-shape code is read.

A character can contain many shapes. You need to decode as many consecutive shape blocks as needed by the character, starting with the shape index in the character definition entry.

**Short Bezier Dictionary**

The short Bezier dictionary is an array of Bezier control points. Each is a triple of 8 + 8 bit coordinates. Each point in the triple is relative to the previous one and the first point of the triple is relative to the current point which you maintain as you decode the shapes.

An entry in this dictionary is accessed through an index referenced by the short Bezier dictionary shape code.

**Long Bezier Dictionary**

The long Bezier dictionary is an array of Bezier control points. Each is a triple of 24-bit packed coordinates. Each point in the triple is relative to the previous one and the first point of the triple is relative to the current point which you maintain as you decode the shapes.

An entry in this dictionary is accessed through an index referenced by the long Bezier dictionary shape code.

**Bitmap Data**

The last section of the font file contains bitmap data referenced in the bitmap directory, if there is any. The first nChars bytes make up the width table of the bitmap, then followed by the pixel map.
stationary, 54
steps for creating, 6–7
system, 44
file, 53
System Log, 141–143
Application Framework, 15
functions, 15
Application Writing Guide, 11–12
Assembly code, viewing, 78–79
ASSERT macro, 134
Asynchronous message passing, 14
ATI video card, 49
ATP subtask, 154
AutoZoom keyword, 36
ZoomMargin and, 40
ZoomResize and, 40
Back menu, 173
bc command, 74
bd command, 92
be command, 92
Bezier curve, 182
edit window and, 184
merging, 186
Bezier dictionaries, 210
bgNc command, 91
Bitmap
changing size of, 173
column major, 195
consistency with outline data, 193
creating, 192–193
Custom Resource ID card, 172–173
data, GO font, 210
deleting, 193
directory, GO font, 207–208
fields, 208editing, 192–195
direction of, 204
at end of, 204
drawing, 182
exporting, 169–170
Exporting option card, 172
to home, 170
illustrated, 170
importing, 168
Bitmap editor, 167–173
bitmap elements, 168
exporting bitmap, 169–170
exporting to home, 170
illustrated, 167
importing bitmap, 168
modes, 168
undo capability, 170–171
user interface, 170–171
Back menu, 173
Document menu, 171
Edit menu, 171
Ink menu, 173
Options menu, 172–173
Size menu, 173
uses, 167–168
Bitmap menu (FEDIT), 192, 193
Edit Cell command, 195
Edit command, 194
BitPad2, sample definition, 62
BitPad2ASCII, sample definition, 62
BkShellPath keyword, 37
bc command, 73
datasheet, 92
BOO1 application process, 154
Boot
error messages, 47
sequence, 29
subdirectory, 28
swap, 39
in tablet-like mode, 41
volume, 29
BOOT.DLC, 42–43
in boot sequence, 29
description, 28
modifying, 42
Booting, 45–49
before, 29
boot error messages, 47
broken pen during, 47–49
loading debug PENPOINT.OS, 46
speeding up, 52
what happens during, 46–47
BootProgressMax keyword, 37
Boo-time install, 52–53
Borders, 16
Bounding rectangle, 181
computation of, 182
control points and, 182
bp command, 73
datasheet, 93
event, 130–131
break command, 79–80
datasheet, 93
Breakpoint, 73–74
commands, 85
direction inside, 131
executing code at, 79–80
profiles, 113, 115
dp command and, 119
zp command and, 121
setting with identifiers, 76
for skipping execution, 123–124
Broken pen icon, 47
INDEX

212

Browser, 19
Built-ins, DB, 125–129
   predefined types, 125–126
   runtime routines, 128–129
   useful values, 126
   useful variables, 127–128
Busy Manager, 19
Buttons, 16
Caches, in running PenPoint, 27
CALC application, 68
CalcEngineEnterOperator
   assembly code, 78–79
   source code, 77–78
CalCompDBII, sample definition, 62
Call stack, viewing, 74–75
   frame numbers and, 75
Capture command (Edit menu), 171
Cartesian grid, 181
Cast operator, 131
C code, executing, 79
   at breakpoint, 79–80
   at DB prompt, 79
   see also C language
Cell dimensions, altering, 195
Character
   adding, 182
   composition, 180–181
   character placement, 181–182
   control point placement, 182
   font units, 181
   data, Nimbus-Q, 205–206
   file positions, 205
   deleting, 182
   dimension alteration, 195
   directory, GO font, 208
   entry structure, 208
   hints, 190
   placement, 181–182
   selection window, 182
   see also Character shapes
Character menu (FEDIT), 179, 182
   adding and deleting characters and, 182
   in changing setwidth, 188
   for character selection window, 182
   Show Sample command, 198
Character shapes, 180–181
   data blocks, 209
   decoding, 210
   data, GO font, 209–210
   definition of, 180–181
   editing, 182–190
   adding oval shape, 187
   adding rectangular shape, 187
   changing setwidth, 188
   deleting segment, 186
   deleting shape, 188
   merging shapes, 189–190
   mitosis, 186
   moving control points, 185–186
   operator icons, 185
   outline editing window, 183–184
   shape mutation, 186
   shape transformation, 187–188
   viewing/altering winding direction, 188
   View Preference, 184–185
   see also Character
CHKDSK, 27
C language, 6, 7
   development options, 5
   reference books, 83
   run-time library, 18
   see also C code, executing
Classes
   layout, 16
   root, 14
   utility, 19
   Class inheritance, 14
   Class Manager, 13–14
   defined, 13
   features supported by, 14
Clipboard. see Windows clipboard
clsByteBuf, 19
clsFileSystem, 19
clsList, 19
clsNotePaper, 19
clsSio, 19
clsStream, 19
clsString, 19
clsTable, 19
cm, co, cs, commands, 94
Code
   addresses, 88, 114
   executable, 14
   profiles, 113
   sample-based, 115
   syntax for, 114
   see also C code, executing; Code profiling
codeAddress, 88
Code profiling, 113–117
   examples, 115–117
   redefining with infinite buckets, 116
   redefining with smaller buckets, 116
   sampling profiles, 115–116
   timing/counting profiles, 116–117
   options, 114
   sampling profiles, 115
   sampling technique, 113–114
   timing/counting technique, 113
   see also Code
Colors
   background, 167
   setting, 173
   foreground, 167
   Invert command and, 171
   setting ink, 173
Command datasheets, 89–112
   <, 90
   ?, 90–91
   !, 89
   ai, 91
   bd, 92
   be, 92
   bgNe, 91
   bl, 92
   bp, 93
   break, 93
   cm, co, cs, 94
   ctx, 94–95
   d, db, dw, dd, 96
   dp, 96
   files, 97
   fl, 97
   fnS, 98
   fs, 98
   g, 99
   h, 99
   id, 99
   ids, 100
   k, 100
   log, 100
   mi, 100
   mini, 101
   od, 101
   on, 101
   on access, on store, 101–102
   p, P, 102
   profile, 103
   q, 103
   r, 103
   srcdir, 103
   st, 104–105
   sym, 105
   ti, 107
   tl, 107
   t, T, 106
   type, 108
   u, 108–109
   uv, 109–110
   v, 110
   vars, 111
   ver, 111
   vu, 111–112
   zp, 112
   see also Commands
Command line editing, 81
Commands
   <, 81
   ?, 75–76, 82, 146
INDEX

.DLC file, 53
DLLMAIN routine, 42, 53
DmIM task, 152
DmKK task, 152
Document, 54
creating new, 54
directories, 55
Documentation, 9–23
feedback on, 10
SDK library, 9
suggested approach to, 9–10
using fonts in, 198–199
see also SDK, documentation
Document menu, 171
Export to Home command, 170, 171
DOS
C development tools, 6, 7
file system utilities, 161–164
GDIR, 162
MAKELABEL, 162–163
APPEND, 163–164
PDEL, 164
PSYNC, 161–162
STAMP, 161–162
LABEL command, 72
name, 162
tools, 13
volume name, 72
dp command, 119–120
datasheet, 96
eamples, 119–120
flags, 119
in redefining profiles, 116
timing/counting profiles, 116
DPrint() function, 67
Drivers
universal pen, 56–62
using UniPen, 60–62
Dynamic Link Libraries (DLL), 20
BOOT.DLC and, 42–43
in boot sequence, 29
Debug versions of, 134
defined, 42
ENVIRON.INI and, 34
files, 53
initialization files and, 28
loaded independent of application, 42, 43

Editing
bitmaps, 192–195
character shapes, 182–190
font header, 196–198
hints, 190
outline, window, 183–184
pixels, 194
Edit menu, 171
Checkpoint command, 170
choosing editing mode from, 171
entries, 171
Embedded objects, 18
Empty Application directory, 52
#endif statement, 134
ENVIRON.INI, 34–40
in boot sequence, 29
DBIni environment, 81
debugger stream and, 135
DebugLogFlushCount line, 136
DebugLog line, 136
DebugSet line, 136
default settings, 34
description, 28
file list of, 35
getting value from, 36
keywords, 36–40
listing of, 36
modifying, 34
setting format, 35
Erase command (Edit menu), 171
Error
codes, 47–49
broken pen error, 1008, 49
brokpen pen errors between 100
and, 999, 49
events, 130
messages, boot, 47
Evaluate command. see ? command
Event handler, values available within, 126
Events
access, 129
bp, 130–131
error, 130
exit, 130
fault, 130
intReq, 130
oc, 131
program, 129
task, 129–130
Examining
font header, 196–198
text samples, 198
Exception handling, 148
Executing
application, 54–55
C code, 79–80
Execution
control commands, 85
controlling threads of, 124
skipping, 123–124
Exit events, 130
Exiting, PenPoint, 55
Exporting
bitmap, 169–170
option card, 172
Export to Home (Document menu), 170
Expression handling (Document menu), 170

Fault events, 130
FEDIT. see Font editor (FEDIT)
Fields
Adobe Type I font, 200
font attribute window, 198
font header window, 197
File commands, 86
File menu (FEDIT), 179, 180
Font Attribute command, 196
Font Header command, 196
Save Subset command, 196
Files
Adobe Type I font, 199
debug stream data, 37
.DLC, 53
DLL, 53
font, editing, 179–180
initialization, 28
map, 148
names of, 52
PENPOINT.DIR, 55
S-Shot, 177–178
swap, 39
error messages and, 47
SYSCOPY.INI, 43
TIFF, 175
used in DB session, 69
WATCOM Make, 148
files command, 97
File system, 18
Fill command (Edit menu), 171nins command, 98
flags
in code profile syntax, 114
dp, 119
in object profile syntax, 117
Flag sets, debugging, 134–135
file command, 97
file formats, 180
Font
adding character to, 193
attribute window, 197
fields, 198
flags
in code profile syntax, 114
dp, 119
in object profile syntax, 117
Flag sets, debugging, 134–135
file command, 97
file formats, 180

Drivers
universal pen, 56–62
using UniPen, 60–62
Dynamic Link Libraries (DLL), 20
BOOT.DLC and, 42–43
in boot sequence, 29
Debug versions of, 134
defined, 42
ENVIRON.INI and, 34
files, 53
initialization files and, 28
loaded independent of application, 42, 43

Editing
bitmaps, 192–195
character shapes, 182–190
font header, 196–198
hints, 190
outline, window, 183–184
pixels, 194
Edit menu, 171
Checkpoint command, 170
choosing editing mode from, 171
entries, 171
Embedded objects, 18
Empty Application directory, 52
#endif statement, 134
ENVIRON.INI, 34–40
in boot sequence, 29
DBIni environment, 81
debugger stream and, 135
DebugLogFlushCount line, 136
DebugLog line, 136
DebugSet line, 136
default settings, 34
description, 28
file list of, 35
getting value from, 36
keywords, 36–40
listing of, 36
modifying, 34
setting format, 35
Erase command (Edit menu), 171
Error
codes, 47–49
broken pen error, 1008, 49
brokpen pen errors between 100
and, 999, 49
events, 130
messages, boot, 47
Evaluate command. see ? command
Event handler, values available within, 126
Events
access, 129
bp, 130–131
error, 130
exit, 130
fault, 130
intReq, 130
oc, 131
program, 129
task, 129–130
Examining
font header, 196–198
text samples, 198
Exception handling, 148
Executing
application, 54–55
C code, 79–80
Execution
control commands, 85
controlling threads of, 124
skipping, 123–124
Exit events, 130
Exiting, PenPoint, 55
Exporting
bitmap, 169–170
option card, 172
Export to Home (Document menu), 170
Expression handling (Document menu), 170
header
editing and examining, 196–198
Nimbus-Q, 205
PenPoint Packed, 206–207
window, 196
window fields, 197
outline, 16
units, 181
using, in documentation, 198–199
Font editor (FEDIT), 179–210
Adobe Type I fonts, 199–204
concepts, 179–182
editing bitmaps, 192–195
editing character shapes, 182–190
effectiveness of, 191
functions, 190
x and y operators, 191, 192
Horizontal Flip command (Edit menu), 171
Hot spot, bitmap, 168
Hotspot Paint mode, 168
Back menu and, 173
defined, 171
Hotspot task, 152
id command, 99
Identifiers
known, 76
scope of, 77
types of, 76
Idle task, 151
ids command, 76, 100
IDS task, 152
#include statement, 134
ImagePoint, 15
imaging model, 15–16
Image shifting, 167
ImageProxy task, 152
Importing, bitmap, 168
Index, master, 13, 21
Initialization files, 28
default versions, 28
Initialization routines, 42
Imgr task, 152
Install, PenPoint, 20
details, 20–21
installer, 20
using, in documentation, 198–199
h command, 99
Header files, debugging flag sets and, 135
Hexadecimal numbers, 72
viewing call stacks and, 74
High-level development tools, 6
development option, 5
Hints, 190
altering, 191
c control handles, 191
creating, 192
data, GO font, 209
deleting, 192
editing, 190–192
window, 190–191
effectiveness of, 191
functions, 190
x and y operators, 191, 192
Horizontal Flip command (Edit menu), 171
Hot spot, bitmap, 168
Hotspot Paint mode, 168
Back menu and, 173
defined, 171
Hotspot task, 152
icommand, 100
Kernel
general protection fault handler, 148
interface, 18
Key task, 152
Keywords
asynchronous serial I/O, 33
ENVIRON.INI, 36
AutoZoom, 36
BkShelfPath, 37
BootProgressMax, 37
Config, 37
DebugLog, 37
DebugLogFlushCount, 37
DebugSet, 38
PenPointPath, 38
PenProxTimeout, 38
ScreenHeight, 39
ScreenWidth, 39
StartApp, 39
StealMem, 39
SwapBoat, 39
SwapFileSize, 39
TZ, 39
Version, 39
VolSel, 40
WinMode, 40
ZoomMargin, 40
ZoomResize, 40
MIL.INI, 33–34
dumping information, 33
files, 33
exit to DOS, 34
high-speed packet parallel port
I/O, 34
k command, 100
Kernel
general protection fault handler, 148
interface, 18
Key task, 152
Keywords
asynchronous serial I/O, 33
ENVIRON.INI, 36
AutoZoom, 36
BkShelfPath, 37
BootProgressMax, 37
Config, 37
DebugLog, 37
DebugLogFlushCount, 37
DebugSet, 38
PenPointPath, 38
PenProxTimeout, 38
ScreenHeight, 39
ScreenWidth, 39
StartApp, 39
StealMem, 39
SwapBoat, 39
SwapFileSize, 39
TZ, 39
Version, 39
VolSel, 40
WinMode, 40
ZoomMargin, 40
ZoomResize, 40
MIL.INI, 33–34
dumping information, 33
files, 33
exit to DOS, 34
high-speed packet parallel port
I/O, 34
serial painting devices, 33
TOPS FlashCard type, 34
video controller, 34
Wacom 310 digitizer, 33

Labelling, volumes, 27–28
Labels, 16
version, 39
volume, 40
LANDSCAPE orientation, 40
Layout classes, 16
lineCount, 88
Line number, 88
Lines, smoothing, 186
List boxes, 16
Loading, system log application, 141
LOCAL keyword, 115
log command, 100
Log file, 135–136
LOGITECH, sample definition, 62
Log Size menu (System Log), 143
Macintosh computer, using S-Shot files on, 177
Macros
debugging, 67
expression handling, 139
message passing, 138
status checking, 134
see also specific macros
MakeWknResld, 173
MAKLABEL utility, 162–163
example, 163
Map files, 148
Mapping, of Adobe standard encoding to AFII codes, 200–204
MAR2 application process, 154
Mask, 168
Mask Paint mode, 168
defined, 171
Memory
available, checking, 81
DB and, 81–82
globally accessible, 129
for running PenPoint, 27
saving, 47
multiple applications and, 53
Usage (Show menu), 142
using less, 39
watching, 82–84
Menu buttons, 16
Menus, 16
Back menu, 173
Document menu, 171
Edit menu, 171
FEDIT
Bitmap menu, 192–195
Character menu, 179, 182, 188
File menu, 179–180, 196
Options menu, 184–185, 191
Outline menu, 183, 195
Text menu, 198
Ink menu, 173
Options menu, 172–173
Size menu, 173
System log application, 142–143
Font menu, 143
Log Size menu, 143
Show menu, 142
Trace menu, 142–143
Merging
shapes, 189
winding direction and, 189–190
Message handling, 14
Message passing
functions, 138
macros, 138
synchronous and asynchronous, 14
Messages
boot error, 47
"Int w/o RB: 7," 47, 50
pattern specifications, 117
sending with DB, 82
string names for, 82
types of, 14
Method table compiler (MT), 165
mi command, 81, 100
MIL.INI, 29–34
in boot sequence, 29
debugger stream and, 135
description, 28
file list of, 30–33
keywords, 33–34
modifying, 29
for digitizing tablet, 45
for mouse, 45
MonoDebug line, 137
setting format, 29
single monitor and, 44
UNIPORT tag, 60
UNIPORTPROTOCOL tag, 61
UNIPEN tag, 60
for writing to serial port, 136
MIL (machine interface layer), 29
during booting, 46
for PC, 29
tablet hardware, 29
mini command, 101
Mini-debugger, 145–154
commands, 146–148
setting debug flags and, 147–148
exception handling, 148
functions, 145
invoking, 145–146
mini-debugger and DB, 145
on PenPoint computer, 146
map files and, 148
source level debugger (DB) and, 145
task list, 151–154
understanding interrupts and, 148–151
using, 148
see also Source level debugger (DB)
Mitosis operator, 186
MM, sample definition, 61
Modifying
BOOT.DLC, 42
ENVIRON.INI, 34
GO.BAT, 45
MIL.INI, 29
Module names, 71
Monitors
application, 21
configurations and, 44
two, 136–137
viewing debugger stream on, 137
writing debugger stream to, 136–137
Mouse
configuring, 45
for running PenPoint, 26
using, 50
Moving, control points, 185–186
msgDcScreenShot, 175
msgList, specification, 118
_MSG_AT variable, 125
Names
DOS, 162
module, 71
process, 71
string, 82
task, 149
volume, 27, 72
Nimbus-Q format, 204–206
AFII number array, 205
character data, 205–206
file positions, 205
converting, 180
FEDIT support of, 180
font header, 205
see also PenPoint, Packed format
Non-maskable interrupt (NMI), 49
Notation conventions, 86–89
code address, 88
data address, 88
line count, 88
line numbers, 88
Scope.Identifier reference, 88–89
scope specification, 86–87
task set, 89
Notebook
default, 46
using, 55
see also specific Notebooks
Notebook User Interface (NUl), two monitors and, 44
Notes, 16
Numbers
frame, 75
hexadecimal, 72
line, 88
ObjectCall, 117
object profiling examples and, 118
objectList specification, 118
Object-oriented programming, 7
Object profiling, 117–118
basic, 117
elements, 118
message pattern, 117–118
options, 117
syntax, 117
Objects
creation of, 14
embedded, 18
observing, 14
profiles, 113
string names for, 82
Observers, 14
od command, 101
on access, on store commands, 101–102
on command, 82–84, 129–131
access events, 129
datasheet, 101
fault events, 130
other events, 130–131
program events, 129
syntax for, 129
task events, 129–130
variations, 82
Operator
cast, 131
Delete, 188
Delete hint, 192
Delete Segment, 186
icons, 185
Merge, 189
mitosis, 186
Move Control Points, 185
Oval, 187
Rectangle, 187
SetWidth, 188
Shape Mutation, 186
Shape Transformation, 187–188
tilde, 131
Winding Direction, 188
x- and y-hint, 191, 192
Option cards, 172
Custom Resource ID, 172–173
Exporting, 172
Option sheets, 17
Options menu, 172–173
FEDIT
AutoRedraw option, 191
BezierResolution command, 184
View Preferences, 184–185
Orientation, screen, 40
OSEnvSearch() function, 36
OSProgramInstall() function, 131
OSProgramInstantiate() function, 131
Outline editing window, 183–184
illustrated, 183
Outline menu (FEDIT), 183, 195
Packed format. see PenPoint, Packed format
Pages, moving through Notebook, 55
Page task, 152
PAPPEND utility, 163–164
example, 164
syntax, 163
Parallel port interrupts, 50
Parity error, 49
Pasting, from Windows clipboard, 195–196
PAUSE key, 71
PCL printer, 199
PC, PenPoint on, 25–62
APP.INI, 44
BOOT.DLC, 42–43
booting, 49
boot sequence, 29
CONSOLE.DLC, 43
desktop and, 55
ENVIRON.INI file, 34–40
executing application, 54–55
hardware specifications for, 25–28
initialization files, setup, 28
installing application, 50–53
MIL.INI, 29–34
PPBOOT.EXE, 29
running, 25–62
stop, 27
setting up specific configurations, 44–45
S-Shot, 178
SYSSAPP.INI, 44
SYSCOPY.INI, 43
in tablet-like mode, 40–42
universal serial pen driver, 56–62
using, 50
PenPoint
exiting, 55
interrupting, 54
invoking mini-debugger on, 146
not working, 47
Packed format, 206–210
AFII number array, 207
bitmap data, 210
bitmap directory, 207–208
character directory, 208
converting, 180
file pointers, 208
font header, 206–207
hint data, 209
long Bezier dictionary, 210
shape data, 209–210
short Bezier dictionary, 210
see also Nimbus-Q format
running on PC, 25–62
PenPoint Architectural Reference, 13–21
Application Framework section, 15
Class Manager section, 13–14
Connectivity section, 20
File System section, 18
Input and Handwriting section, 17
Installation API section, 20–21
Resources section, 20
System Services section, 18
Text component section, 17–18
UI Toolkit section, 16–17
Utility Classes section, 19
Writing PenPoint Services section, 21
PenPointPath keyword, 38
PenPoint UI Design Reference, 13
PenProxTimeout keyword, 38
Pen task, 152
Pixelmaps, 167
Pixel Paint mode, 168
Back menu and, 173
defined, 171
Ink menu and, 173
Pixels, 168
editing, 194
metrics, 192
PORTRAIT, screen device orientation, 40
Ports
parallel, interrupts, 50
serial, 34
writing to, 136
PostScript
interpreter, 199
printer, 198
INDEX

Powr task, 152
PPBOOT boot program, 29
during booting, 46–47
p (P) commands, 77
datasheet, 102
Preferences Power option sheet, 27
Process
commands, 85
names, 71
processCount, 153
profile command, 103
Preferences Power option sheet, 27
Process
commands, 85
names, 71
processCount, 153
profile command, 103
Profiles
breakpoint, 113
clearing, information, 121
code, 113
displaying, information, 119–120
object, 113
redefining with infinite buckets, 116
redefining with smaller buckets, 116
samples of, 115
timing/counting, 116–117
types of, 113
Profiling
code, 113–117
commands, 86
object, 117–118
specific tasks, 121
Program
events, 129
using DEBUG in, 134
Programmable interrupt controller
(PIC), 50
Progress bars, 17
Project Scheduler, 53
Proximity, out of, 38
mouse and, 50
Q
q command, 84
datasheet, 103
Quick Help, 19
Quick Installer, 50
RAM
disks, 27
for running PenPoint, 25, 27, 54
r command, 103
RC utility, 165
Rectangle
adding, shape, 187
bounding, 181–182
Reference, Scope.identifier, 88–89
Remote interface, 20
RESAPPND utility, 165
bitmap resources and, 169
RESDUMP utility, 165
Resource, 20
files, 165
application, 169
ID, 169
Custom Resource ID card, 172–173
utilities, 165
Root classes, 14
Rotate command (Edit menu), 171
Routines
DB runtime, 128–129
skipping execution of, 123–124
routineSet specification, 114
_ROUTINE_SET variable, 125
Running
PenPoint on PC, 25–62
system log application, 141–142
Run-time libraries, 18
Sample
applications, 12
profiles, 115
Sampled images, 167
Saving
Adobe font, 199–200
characters and bitmaps subset, 196
typing, 80–81
Scav task, 152
Scope
redefining, 88–89
specification, 86–87
Scope.Identifier specification, 77
scopeSpec, 86–87
Screen
capturing, 175–178
before, 176
ScreenHeight keyword, 39
ScreenWidth keyword, 39
Scroll bars, 16
SDK
contents, 22–23
compiler tools, 23
optional goodies, 23
PenPoint, 22
documentation
API Reference, 21
Application Writing Guide, 11–12
Architectural Reference, 13–21
Development Tools, 13
documents in, 10
feedback on, 10
library, 9
organization of, 13
sample applications in, 12
suggested approach to, 9–10
UI Design Reference, 13
see also Documentation
Search and Replace API, 19
Segments, 186
deleting, 186
shape mutation and, 186
Selection Manager, 19
Serial ports, 34
writing to, 136
Services Architecture, 21
Settings notebook
Gesture Timeout, 38
Installer, 41
Setwidth, 180
changing, 188
character placement and, 181
Shapes. see Character shapes
Shift command (Edit menu), 171
Show menu (System Log), 142
Single-stepping, 77
Size menu, 173
SmartDrive, 27
Snapshots, 175
file name, 176
full-screen, 177
doing, 177
impossible, 177
taking, 177
before, 176
writing to disk, 177
Source code
debugging, 72–73
setting breakpoint and, 74
viewing, 77–78
Source level debugger (DB), 67
activation of, 84
advanced techniques, 123–131
cast operator, 131
compile time commands, 124
context inside breakpoints, 131
controlling execution threads, 124
install and start commands, 131
on command, 129–131
skipping execution, 123–124
tilde operator, 131
wait command, 131
built-ins, 125–129
predefined types, 125–126
runtime routines, 128–129
useful values, 126
useful variables, 127–128
command line editing, 81
command reference, 85–112
command datasheets, 89–112
command summary, 85–86
notation conventions, 86–89
compiling and linking and, 69–70
context, 73
DB.INI file, 80–81
using DB scripts, 81
TZ keyword, 39
u command, 78–79
datasheet, 108–109
UI Toolkit, 16–17
Undo Manager, 19
UniPen, 56
command syntax, 57–59
notes on using, 60–62
sample definitions, 61–62
UNIPENPORT tag (MILINI), 60
UNIPENPROTOCOL tag (MILINI), 61
UNIPETYPE tag (MILINI), 60
predefined types, 60
Unique identifier (UID), 14
dynamic, 14
numbers, 14
well known, 14
Universal serial pen driver, 56–62
UniPen command, 57–59
using, 60–62
User interface, bitmap editor, 170–173
Utilities
GDIR, 162
GO, 165
MAKLABEL, 162–163
MT, 165
PAPPEND, 163–164
PDEL, 164
PSYNC, 164
RC, 165
RESAPPND, 165
RESDUMP, 165
S-Shot, 175–178
STAMP, 161–162
Utility classes, 19
uv command, 109–110
V86a task, 152
V86x task, 152
Values
examining and setting, 75–77
? command, 75–76
identifier types, 76
known identifiers, 76
lexical scope, 77
useful, in DB, 126
Variables
DB useful, 127–128
debugging flag sets, 134–135
setting, 127
vars command, 111
v command, 77–78
datasheet, 110
ver command, 111
Version keyword, 39
Vertical Flip command (Edit menu), 171
VGA video adapter, 25
View Preference dialog (FEDIT Options menu), 184–185
illustrated, 185
VolSel keyword, 40
WACOM510, sample definition, 62
WACOM510C, sample definition, 62
Wacom pen tablet, 45
wait command, 131
WATCOM
C/386 compiler and linker, 69–70
Make files, 148
Winding direction, 188
merging and, 189–190
viewing and altering, 188
Window
bitmap edit, 194
color selection, 182
text attribute, 197
font header, 196
hint editing, 190–191
illustrated, 191
outline editing, 183–184
system, 15
Windows clipboard
copying to, 195
pasting from, 195–196
WinMode keyword, 40
Writing
to debugger stream, 137–139
to log file, 135–136
to second monitor, 136–137
to serial port, 136
ZoomMargin keyword, 40
AutoZoom and, 36
ZoomResize keyword, 40
AutoZoom and, 36
zp command, 121
datasheet, 112
This master index indexes all five volumes in the PenPoint Software Development Kit documentation. Each page number in the master index contains a code that indicates the volume in which the page is found. The codes are:

**AWG**  PenPoint Application Writing Guide
**AR1**  PenPoint Architectural Reference Volume I
**AR2**  PenPoint Architectural Reference Volume 2
**UI**  PenPoint User Interface Design Reference
**PDT**  PenPoint Development Tools

### 3-D icons, UI:217
- 16-bit character
  - string functions, AR2:111–114
  - composition, AR2:114
  - support, AR2:110–114
  - features, AR2:110
  - types, AR2:111
- 8259 programmable interrupt controller (PIC), AR2:275

### 80386
- protected mode, AR2:102
- ring structure, AR2:103

### ? command, PDT:75–76, 82
data sheet, PDT:90–91
for displaying mini-debugger commands, PDT:146

### ! command, PDT:79
data sheet, PDT:89

### < command, PDT:81
data sheet, PDT:90

### AB_MGR_ID structure, AR2:329
**AB_MGR_NOTIFY structure, AR2:330**

### About Application sheet, UI:61
- customizing, UI:194

### About command (Document menu), UI:57, 61

### About Contents command (table of contents Document menu), UI:85

### About Document sheet, UI:61

### Abs macro, AWG:78

### Abstract messages, AWG:57

### Access
- events, PDT:102
- intentions, AR2:62
- protocols, AR2:257
  - sheet, UI:67, 201
  - Access Speed control, UI:67
  - active documents and, UI:102
  - customizing, UI:201
- Speed, AWG:39

### Accessing services, AR2:258–259
  - binding to a service, AR2:259
  - opening service, AR2:259
  - service managers, AR2:261
  - predefined, AR2:258–259
  - see also Services

### Accessories
  - UI:163; AR2:393
documents, AR2:377
  - icon, UI:13, 74
  - application, UI:217
  - palette, AWG:104, 134
  - icons in, UI:73, 163, 217
  - window, AWG:23
  - see also Floating, accessories

### Accessories notebook, S-Shot, PDT:175

### Accessory
  - icons, UI:77
  - instances, UI:163
  - keyboard, UI:202, 273

### ACCESSORY directory, AR2:393

### Acetate layer, AR1:296

### Acetate plane, AWG:9

### Activating
  - documents, AWG:23, 36–37;
  - AR1:102–107
  - embedded documents, AR1:161
  - fields, AR1:481–482

### Active In box service, AR2:312

### ADC labs, AWG:195

### Adder application, AWG:260–261;
  - AR1:556

### Adding
  - address book entry, AR2:328
  - characters, PDT:182
  - document to stationary menu, AR2:426
  - items to transaction, AR2:203
  - list items, AR2:129
  - network protocols, AR2:251
  - oval shape, PDT:187
  - rectangle shape, PDT:187
  - rows to tables, AR2:222–223

### transfer types, AR2:173
see also Installing

### ADDR_BOOK_ATTR structure, AR2:320–321

### ADDR_BOOK_ENTRY structure, AR2:322, 327
allocation of, AR2:328

### ADDR_BOOK_QUERY_ATTR structure, AR2:326–327

### AddrBookStreetId, address book identifier, AR2:321

### Address book, AWG:40; AR2:241, 317–330
  - changing information in, AR2:328
  - closing, AR2:326
  - concepts, AR2:317
  - defined, AR2:317–318
  - entry
  - adding, AR2:328
  - attribute identifiers, AR2:321
  - attributes, AR2:320–322
  - deleting, AR2:328
  - groups, AR2:322
  - organization, AR2:320
  - service addresses, AR2:322
  - system, AR2:329–330
  - sendable services protocol uses, AR2:331
  - system, AR2:329–330
  - deactivating, AR2:330
  - defined, AR2:318
  - theAddressBookMgr and, AR2:318
  - unregistering, AR2:329
  - using, AR2:325–328
  - writing, AR2:328–330

### Address Book icon, UI:77
Address book manager protocol, AR2:320
  function, AR2:318–319
Address book protocol, AR2:319
  function, AR2:318
Address descriptors, AR2:331–332
  getting, AR2:333
Addresses
  code, PDT:88
  data, PDT:88
Address List, AWG:13
Address window, AR2:332
  creating, AR2:333
  filling, AR2:333–334
Add structure, AWG:46–47
Adobe Type I font, PDT:199–204
  FEDIT support of, PDT:180
  fields, PDT:200
  saving, PDT:199–200
  standard encoding to AFII mappings, PDT:200–204
Adobe Type Manager, PDT:199
Advisory messages, AWG:57
AFII number, PDT:180
  altering, PDT:182
  GO font array, PDT:207
  Nimbus-Q array, PDT:205
Agents, resource, AR2:345
  ai command, PDT:91
Alarm services, AR2:103
Align commands (MiniNote Arrange menu), UI:137
Aligning
  constraints for, AR1:393
  width and height dimensions, AR1:393–394
Altering
  hints, PDT:191
  winding direction, PDT:188
Ancestor class, AWG:43–44
  CLASS_NEW message argument and, AWG:54
  initializations and, AWG:118–119
  message handling and, AWG:44
  messages, AWG:60
  _NEW_ONLY structure and, AWG:50–51
  self UIDs and, AWG:56
  see also Classes
Ancestors, AR1:5–6
  calls, AR1:36–37
  CLASS_NEW_ONLY structure, AR1:47
  confirming object, AR1:55
  gestures and, UI:233
  inheritance and, AR1:82
  mark component, AR1:201
  toolkit, AR1:367–370
ANM_CREATE_DOC structure, AR2:425
ANM_CREATE_SECT structure, AR2:424
ANM_DELETE_ALL structure, AR2:426
ANM_DELETE structure, AR2:426
ANM_GET_NOTEBOOK structure, AR2:423–424
ANM_MOVE_COPY_DOC structure, AR2:425–426
API
  installation, PDT:20–21
  Quick Help, PDT:19
  search and replace, PDT:19
API Reference, PDT:21
APP_ACTIVATE_CHILD structure, AR1:161
appAttrClass, AR2:148
APP directory, AR2:384, 386
  contents, AR2:386
  directory contents, AR2:386–387
APP_DIR_GET_BOOKMARK structure, AR1:182
APP_DIR_GET_SET_ATTRS structure, AR1:180
APP_DIR_GET_SET_FLAGS structure, AR1:180
APP_DIR_NEXT structure, AR1:182
APP_DIR_SEQ_TO_NAME structure, AR1:183
APP_DIR_SET_BOOKMARK structure, AR1:182, 183–184
APP_DIR_UPDATE_CLASS structure, AR1:181
APP_DIR_UPDATE_NUM_CHILDREN structure, AR1:181–182
APP_DIR_UPDATESEQUENCE, AR1:181
APP_DIR_UPDATE_UUID structure, AR1:181
APP_DIR_UUID_TO_NAME structure, AR1:183
APP_EXECUTE structure, AR1:165–166
APP_GET_GLOBAL_SEQUENCE structure, AR1:179
APP.INI, AR2:387; PDT:44
  in boot sequence, PDT:29
  in boot-time install, PDT:52–53
  description, PDT:28
  service directory and, AR2:444
AppleTalk, AWG:8
  protocol, AR2:301–302
  changing size of ATP packets and, AR2:302
  name, AR2:302–304
  options, AR2:301–302
  zone, AR2:304
  services, AR2:250
AppleTalk transport protocol (ATP), AR2:253, 297
changing packet size, AR2:302
Application classes, AWG:22, 26; AR1:69, 82, 157–171
advanced messages, AR1:171
clsApp messages, AR1:157–161
clsHelloWorld, AWG:125
creating, AWG:103
  defined, AR1:67
document hierarchy messages, AR1:161–163
document window messages, AR1:163
  efficiency, AR1:85
embedded documents, AR1:77
Empty Application, AWG:88
  getting and setting, AR1:181
  initialization routine, AR1:96, 97–99
  instance of, AR1:67, 92
  instances and, AWG:24
  life cycle, AR1:95–99
  deinstalling application, AR1:99
  installing application, AR1:96–99
  messages, AR1:157–161
  received by, AR1:99
  method tables and, AWG:99
  observing system preferences, AR1:170
  PenPoint process and, AWG:104
  processes, AR1:90–91
  relationships of, AWG:29
  sections and, AWG:34
  standard application menus, AR1:163–170
  state diagram, AR1:95
  states, AR1:96
  well known, AWG:104
  see also Classes; clsApp
Application design. see Design, guidelines
Application development. see Development
Application directories, AR1:177;
  AR2:391–395
  accessories, AR2:393
  attributes, AR1:180–182
  many, AR1:182–183
  creating, AR2:391
  files in, AR2:391–392
  flags, AR1:180
  global data, AR2:394–395
  global sequence number, AR1:179
  handle, AR1:177
  creating, AR1:179
  destroying, AR1:179
  help, AR2:393–394
  stationary, AR2:392–393
  see also Applications
Application directory handle class, AR1:70, 177–184
  counting embedded documents, AR1:183
  creating directory handle, AR1:179
  destroying directory handle, AR1:179
  directory attributes, AR1:180–182
Applications page (Installed Software section), UI:98 views, UI:109 option menu, UI:111

Application window changing style of, AR1:196 closing, AR1:196 metrics of, AR1:196–197 opening, AR1:196

Application window class, AR1:195–197 see also clsAppWin

Application Writing Guide, PDT:11–12

Apply button
clean and dirty controls and, UI:46–47 option sheets and, UI:44

AppMainO function, AWG:25, 106; AR1:72 activating application and, AWG:107 document activation and, AR1:102, 105

AppMain routine, AR1:97

APP_METRICS structure, AR1:161–162

APP_MGR_ACTIVATE structure, AR1:148

APP_MGR_CREATE structure, AR1:148

APP_MGR_DELETE structure, AR1:149

APP_MGR_METRICS structure, AR1:146–147, 149

APP_MGR_MOVE_COPY structure, AR1:148–149

APP_MGR_NEW structure, AR1:97

AppMonitorMain() function, AWG:105, 107; AR1:72, 97, 151;

AR2:377–378

in installation, AR1:99

appProcess parameter, AR1:96

APP.RES, AWG:168

application message resource, AWG:169

creating icons, AWG:172

APP_SET_PRIORITY structure, AR1:171

APP_WIN_METRICS structure, AR1:196–197

APP_WIN_NEWONLY structure, AR1:196

Arc, AR1:271

Architecture
functionality, AWG:6

object-oriented, AWG:5

argc parameter, AR1:96

Argument data, AR1:14

modifying, AR1:17

Arguments, AWG:99

Argument structure, elements, AWG:50

argy parameter, AR1:96

Arrange menu (MiniNote), UI:137

Arrows & Drag Box scroll margins, UT:263–264

Arrows, for scroll margin, UI:36

Arrows gesture family, UI:24

Arrow (Up and Down) gesture, UI:25

Double, UI:266

guidelines for, UI:240

MiniText, UI:133

for zooming, UI:266

ASCII, UI:289

metrics transfer, AR2:175
text file creation, AR2:180

Assembly code, viewing, PDT:78–79

Assertions, AWG:85

ASSERT macro, AWG:85; PDT:134

Asynchronous message passing, PDT:14

AT command, modem, AR2:286–287

set, AR2:290–293

ATI video card, PDT:49

At-least-once-delivery, datagram, AR2:296

Atom identifier, AR2:15

Atoms, AR2:37–38
defined, AR2:37

for nil string, AR2:37

predefined, AR2:38

ATP_OPTIONS structure, AR2:301–302

ATP subtask, PDT:154

Attributes, AR2:7–9

directory book entry, AR2:320–322

identifiers, AR2:321

arguments, AR2:16–18

changing, AR2:8, 19–20

character, AR2:8–9, 16–17

clearing, AR2:20

default, AR2:7

file system, AR2:55, 77

client defined, AR2:77–78

getting and setting, AR2:76–80

getting values, AR2:78

length of values, AR2:79

lits of, AR2:76–77

setting values, AR2:79

zero value, AR2:77

getting and setting, AR2:16–20

initializing, AR2:19

installable item, AR2:412

label macros, AR2:77–78

local, AR2:7

modifying, AR2:19–20

node, AR2:54–55

client-defined, AR2:54

file system, AR2:55

flags, AR2:79–80

paragraph, AR2:2, 17–18

type values, AR2:76

Audible feedback, UI:217–218

default, UI:213

for minor errors, UI:213–214
Auto-answer mode, modem, AR2:284
Automatic layout (User Interface Toolkit), AWG:10
Auto Power-Off Devices (Power preferences), UI:96
Auto-selection, UI:283
Auto Shutdown (Power preferences), UI:96, 97
Auto shutdown preference, AR2:365
Auto Suspend (Power preferences), UI:96, 97
Auto suspend preference, AR2:364
AutoZoom keyword, P、DT:40
ZoomMargin and, P、DT:40
ZoomResize and, P、DT:40
Auxiliary notebook manager, AR2:421-427
messages, AR2:423
generalized, AR2:423-424
specialized, AR2:424-426
Auxiliary notebooks, AR2:380, 421-422
back up considerations, AR2:422
creating, AR2:421-422
creating, sections, AR2:424
deleting section/document, AR2:426
file system and, AR2:422
going paths to, AR2:423-424
list of, AR2:422
moving/copying documents to, AR2:425-426
opening, AR2:423
tags, AR2:421-422
see also Notebook

Bell preference, AR2:365
Bezier curve, AR1:271; P、DT:192, 193
edit window and, P、DT:184
merging, P、DT:186
Bezier dictionaries, P、DT:210
B gesture, UI:27
in gesture mode, UI:259
in MiniNote, UI:142
in MiniText, UI:134
bgNc command, P、DT:91
Binding, AR1:210; AR2:247
to local transport address, AR2:301
to service, AR2:259
Bit manipulation, AWG:79
Bitmap, AWG:171-172; AR1:329-331
changing size of, P、DT:173
column major, P、DT:195
consistency with outline data, P、DT:193
creating, AR1:330; P、DT:192-193
Custom Resource ID card,
P、DT:172-173
data, GO font, P、DT:210
deleting, P、DT:193
directory, GO font, P、DT:207-208
fields, P、DT:208
editing, P、DT:192-195
directory, GO font, P、DT:207-208
edit window, P、DT:194
elements, P、DT:168
exporting, P、DT:169-170
Exporting option card, P、DT:172
to home, P、DT:170
illustrated, P、DT:170
for icons, UI:217-218
small and large, UI:220
images, AR1:263
allocating, AR1:258
fonts and, AR1:314
importing, P、DT:168
messages, AR1:329-330
modifying, AR1:330
notifications, AR1:331
pictures, AR1:524-525
tags, AR1:330
using, AR1:330
Bitmap editor, P、DT:167-173
bitmap elements, P、DT:168
exporting bitmap, P、DT:169-170
exporting to home, P、DT:170
illustrated, P、DT:167
importing bitmap, P、DT:168
modes, P、DT:168
undo capability, P、DT:170-171
user interface, P、DT:170-173
Back menu, P、DT:173
Document menu, P、DT:171
Edit menu, P、DT:171
Ink menu, P、DT:173
Options menu, P、DT:172-173
Size menu, P、DT:173
uses, P、DT:167-168
Bitmap editor icon, UI:77, 216
Bitmap menu (FEDIT), P、DT:192, 193
Edit Cell command, P、DT:195
Edit command, P、DT:194
BITMAP_NEW structure, AR1:330
BitPad2, sample definition, P、DT:62
BitPad2ASCII, sample definition, P、DT:62
BkShellPath keyword, P、DT:37
bl command, P、DT:73
datasheet, P、DT:92
Block, AR2:7
Blocking protocol, AR2:168-169
deadlocks, AR2:169
defined, AR2:168
Bold buttons, UI:28, 177
Bold style, UI:16
BOO1 application process, P、DT:154
Bookshelf, AWG:9; UI:13-14
Accessories icon, AWG:95
disk-based, UI:13-14
icon option sheet and, UI:75
icons, UI:13, 74
closed and open states of,
UI:221-222
list of, UI:76
illustrated, UI:12
main, UI:13
reference buttons on, UI:20, 171
targeting policy, UI:233
view, UI:109
Layout sheet, UI:112
Boolean operators, table, AR2:225
BOOLEAN type, AWG:77
Boot
disks, services on, AR2:444
error messages, P、DT:47
progress messages, AR2:431-432
sequence, AR2:429; P、DT:29
symbols, AR2:429
subdirectory, P、DT:28
swap, P、DT:39
in tablet-like mode, P、DT:41
volume, AR2:43; P、DT:29
BOOT directory, AR2:384
structure, AR2:385
BOOT.DLC, P、DT:42-43
in boot sequence, P、DT:29
description, P、DT:28
modifying, P、DT:42
Booting, P、DT:45-49
before, P、DT:29
boot error messages, P、DT:47
broken pen during, P、DT:47-49
loading debug PENPOINT.OS, P、DT:46
speeding up, P、DT:52
what happens during, P、DT:46-47
BootProgressMax keyword, PDT:37
Boot time, AWG:94
install, PDT:52–53

BORDERBACKGROUND structure, AR1:377
BORDER_NEW structure, AR1:373
styles, AR1:373–375
Border rectangle region, AR1:379
Borders, AR1:361; PDT:16
adjusting, AR1:505
button, UI:177
for embedded documents, UI:19–20
geometry, AR1:379–380
outer offsets, AR1:380
subclassing clsBorder and, AR1:380
toolkit ancestors and, AR1:370

see also Border windows

BORDERSTYLE structure, AR1:373

Border windows, AR1:371–380
bsUnitsLayout measurement, AR1:375
creating, AR1:373–375
going, AR1:379–380
layout, AR1:378
messages for, AR1:371–373
painting, AR1:375–377
propagating/notifying visuals and, AR1:378
regions of, AR1:379
resizing, dragging, topping, AR1:377
sample, AR1:371
visuals of, AR1:376

see also clsBorder

Bounding box, AR1:397
for drag feedback, UI:280
Bounding rectangle, PDT:181
computation of, PDT:182
control points and, PDT:182

Bounds, AR1:232
accumulation, AR1:270–271
defined, AR1:269
client interface to layout, AR1:249
DC, AR1:256–257
scribble, AR1:607
setting, AR1:234

Boxed lists, UI:32
checklist vs., UI:182–183
illustrated, UI:32
multiple, UI:184
choice, UI:32
modes, UI:248
palette line with, UI:39

Boxed pads, UI:50
buttons, UI:50
writing, UI:48
embedded, UI:49
pop-up, UI:48
using, UI:51

bp command, PDT:73
datasheet, PDT:93

event, PDT:130–131
Brackets gesture, UI:25
adjusting list selection with, UI:281
adjusting text selection with, UI:281
extending selection with, UI:280–281
family, UI:24
in gesture mode, UI:259
guidelines for, UI:238
hot point for, UI:231
MiniNote, UI:141

see also Gestures
break command, PDT:79–80
datasheet, PDT:93
Breakpoint, PDT:73–74
commands, PDT:85
code inside, PDT:131
executing code at, PDT:79–80
profiles, PDT:113, 115
dp command and, PDT:119
zp command and, PDT:121
setting with identifiers, PDT:76
for skipping execution, PDT:123–124
BREAK signal, AR2:272
Bring To command (table of contents View menu), UI:86
Broken pen icon, PDT:47

Browser, AR2:124; PDT:19
changing, client, AR2:144
class, AR2:137–145
customizing, UI:177–178
defaults, AR1:419
delay and reference count, AR2:194
display, UI:81
manager, AR2:193–194; PDT:19
function, AR2:124
Button definition, AR1:495

BUTTONMETRICS structure, AR1:423

BUTTONNEW_ONLY structure, AR1:419

Buttons, UI:28–29; AR1:417–424; PDT:16
choice component, AR1:362
command, UI:178; AR1:495
defined, UI:179
non-standard, UI:179
standard modal, UI:178–179
standard modeless, UI:178
creating, AR1:354, 419–420, 421
many, AR1:422
customizing, UI:177–178
defaults, AR1:419
types of, AR1:418
labels for, UI:175
deactivating, UI:175
wording, UI:210–211
manager, AR1:423
messages, AR1:417–418
non-command, UI:179
notification, AR1:420–421
advanced techniques, AR1:423–424
unwelcome, AR1:421

on boxed pads, UI:50
painting, AR1:422
preview messages, AR1:362
sample, AR1:417
showDirty control, AR1:422
styles for, UI:175–178
bold, UI:177
half-outlined, UI:177
outlined, UI:176
raised, UI:176
square, UI:176
styles of, AR1:418, 419–420
types of, UI:28–29
in user interface, UI:151
value, AR1:422
Cached image, AR1:273-274, 299-301
creating, AR1:300
defined, AR1:299
drawing, AR1:300-301
hot spot, AR1:273, 300
image devices vs., AR1:301
invalidating, AR1:301
mask, AR1:300, 301
related classes, AR1:301
Caches, in running PenPoint, PDT:27
CALC application, PDT:68
CalcEngineEnterOperator
assembly code, PDT:78-79
source code, PDT:77-78
CalCompDBII, sample definition, PDT:62
Calculator, AWG:261-262; UI:13
Callback function, AR1:273
sampled images and, AR1:299
Call stack, viewing, PDT:74-75
frame numbers and, PDT:75
Cancel button
command, UI:178-179
for progress notes, UI:211
Canvas, AR1:215
Capabilities, object, AR1:25-29
changing capability, AR1:29
checking capability, AR1:29
creation capabilities, AR1:28
creation notification, AR1:28
flags, AR1:25
freeing capability, AR1:26
inheritance capability, AR1:27
mutation capability, AR1:28
ObjectCall() capability, AR1:26
ObjectSend() capability, AR1:26
observable capability, AR1:27
owner capability, AR1:26
scavenging capability, AR1:27
Cap argument, AR1:24
changing capabilities and, AR1:29
OBJECT_NEW_.structure, AR1:47
Capitalization convention
in button labels, UI:175
for messages, UI:210
Capital letter gestures, UI:27
collisions and, UI:233-234
hot point for, UI:231
see also Gesture accelerators
Capture command (Edit menu), PDT:171
Caret gesture, UI:16, 24
for Create menu, UI:87
family, UI:23-24
summary, UI:235
in gesture mode, UI:258
guidelines for, UI:237
MiniNote, UI:141
tab stops and, UI:132
see also Gestures
Caret-Tap gesture, UI:25
guidelines for, UI:239
MiniNote, UI:140
MiniText, UI:133
see also Gestures
Carrier state, modem, AR2:284
Cartesian grid, PDT:181
Cast operator, PDT:131
C code, executing, PDT:79
at breakpoint, PDT:79-80
at DB prompt, PDT:79
see also C language
Cells
altering dimensions of, PDT:195
reference button labels for, UI:172
Center command (MiniNote Arrange menu), UI:137
C gesture, UI:27, 56
Changing capability, AR1:29
CHAR8, AR2:111
CHAR16, AR2:111
CHAR, AR2:111
Character
adding, PDT:182
box height preference, AR2:366
box width preference, AR2:366
composition, PDT:180-181
character placement, PDT:181-182
control point placement, PDT:182
font units, PDT:181
data, Nimbus-Q, PDT:205-206
file positions, PDT:205
deleting, PDT:182
dimension alteration, PDT:195
directory, GO font, PDT:208
entry structure, PDT:208
hints, PDT:190
lists template, UI:242
metrics, AR1:310
page, UI:46
placement, PDT:181-182
positions, AR1:416
recognition engine, UI:234
selection window, PDT:182
types, AWG:62
string constants and, AWG:62-63
widths, AR1:309-310
see also Characters, 16-bit; Character shapes
Character menu (FEDIT), PDT:179, 182
adding and deleting characters and, PDT:182
in changing setwidth, PDT:188
for character selection window, PDT:182
Show Sample command, PDT:198
Character option sheet (MiniText), UI:131
Characters
16-bit, AR2:110-114
attributes of, AR2:8-9, 16-17
deleting, AR2:11
font masks, AR2:17
getting range of, AR2:14
getting single, AR2:14
inserting, AR2:12
reading, in text data objects, AR2:14
scanning ranges of, AR2:15-16
types of, AR2:111
Character shapes, PDT:180-181
data blocks, PDT:209
decoding, PDT:210
data, GO font, PDT:209-210
definition of, PDT:180-181
editing, PDT:182-190
adding oval shape, PDT:187
adding rectangular shape, PDT:187
changing setwidth, PDT:188
deleting segment, PDT:186
deleting shape, PDT:188
merging shapes, PDT:189-190
mitosis, PDT:186
moving control points,
PDT:185-186
operator icons, PDT:185
outline editing window,
PDT:183-184
shape mutation, PDT:186
shape transformation,
PDT:187-188
viewing/altering winding direction,
PDT:188
View Preference, PDT:184-185
see also Character
CHAR types, AWG:62-63, 77
versioning data and, AWG:63
Checkboxes, UI:32
multiple, UI:184–185
for scrolling multiple checklists, UI:187

Check gesture, UI:16, 24
Connected Disks page, UI:115
in gesture mode, UI:258
guidelines for, UI:237
handling, AR1:169–170
hot point for, UI:231
MiniNote, UI:135, 141
on document title, UI:89
option sheets response to, UI:203
over icons, UI:75
processing, AR1:517–521
card client activity, AR1:519–520
dimmed controls, AR1:521
mixed attributes, AR1:520
multiple card types, AR1:520
multiple option sheets, AR1:521
nested components, AR1:521
run-through, AR1:517–519
selection interaction, AR1:521
response to, UI:46
see also Gestures

Checking capability, AR1:29

Checklists, AWG:68; UI:29
alternative
boxed lists, UI:32
toggle switch, UI:30
boxed lists vs., UI:182–183
design, UI:295–296
with fields, UI:186
menus with, UI:41–42
multiple, UI:31, 184
comparison with single, UI:31
scrolling, UI:187
of non-essential items, AWG:69
pop-up, UI:30
scrolling, UI:196
of required interactions, AWG:68–69
scrolling, UI:196

Checkmark, UI:182
gesture mode picture, UI:248
in zero or one style, UI:185

Checkpoint command (Document menu), UI:57, 193

Checks gesture family, UI:24
summary, UI:235
see also Check gesture; Check-Tap gesture

Check-Tap gesture, UI:25
guidelines for, UI:239

Child windows, AR1:216
altering, AR1:246
labels and, AR1:415–416
layout, AR1:415–416
painting, AR1:416
toolkit tables, creating, AR1:434

CHKDSK, PDT:27
Chk() macros, AR1:24
Choice, AR1:427, 442–444
component buttons, AR1:362
creating, AR1:352–353, 443
management, AR1:439–440
selection, AR1:440
manager, AR1:443
messages, AR1:443
notification, AR1:443
value, AR1:443–444

CHOICE_NEW structure, AR1:443, 444

Chord figure, AR1:272
Circle-Flick-Down gesture, UI:25
Circle-Flick gesture (Up and Down), UI:133
Circle-Flick-Up gesture, UI:25
Circle gesture, UI:16, 24
ingoing pads and, UI:189
family, UI:24, 235
gesture mode and, UI:257, 258, 259
guidelines for, UI:237
hot point for, UI:231–232
MiniNote gesture mode, UI:135, 141–142
MiniNote ink mode, UI:135
MiniText, UI:133
see also Gestures

Circle-Line gesture, UI:25
gesture mode and, UI:257, 259
guidelines for, UI:239
Circle-Tap gesture, UI:25
gesture mode and, UI:257, 259
guidelines for, UI:239

CAlign() macro, AR1:393

C language, PDT:6, 7
code file, AR1:31
compiler
portability, AWG:76
unused parameters and, AWG:110
for defining resources, AR2:355–357
development options, PDT:5
function calls, AWG:67
programming, AWG:19
reference books, PDT:83
runtime, AWG:7
run-time library, PDT:18
source, AWG:67
code, AWG:98
see also C code; C code, executing

Class
browser, AWG:118
counter, AWG:138
creation, AWG:103–104
hierarchy, AWG:44
looking up, AWG:119
info table, AWG:55
names, AWG:71, 79

UID, AWG:102–103
value, AR1:55

Classes, AWG:20, 41; AR1:5

Adder, AWG:261
application, AR1:82, 86–87
clsAppMgr and, AR1:85
container, AR1:185–188
inheritance of, AR1:86
initialization routine, AR1:96, 97–99
life cycle of, AR1:95–99
processes of, AR1:90–91
state diagram, AR1:95
states of, AR1:96
summary, AR1:87

Application Framework, AR1:68–70
hierarchy, AR1:71
applications and, AWG:24
mix of, AWG:30
Basic Service, AWG:272
built-in, AWG:116
Calculator, AWG:262
class info table and, AWG:55
class’s, getting, AR1:55
Clock, AWG:263–264
code sharing, instead of, AWG:43–45
compiling, AR1:47
components, AWG:66; AR1:76
Counter Application, AWG:196
creating, AWG:53–58; AR1:6, 31–48
ancestor calls and, AR1:36–37
design considerations, AR1:33
header file and, AR1:36
subclassing, AR2:67
defining, AR1:5
definition, AWG:125
design guidelines and, AWG:16
designing, AWG:60
drawing contexts, AWG:128
Empty Application, AWG:177
extending existing, AWG:44
file system, AR2:62

subclassing, AR2:67
getting information about, AR1:54–56
handwriting
capture, AR1:551
translation, AR1:552–553
Hello World (custom window), AWG:187
Hello World (Toolkit), AWG:181
implementation of, AR1:32
Inputapp, AWG:270
installation, AR2:379–380
Installer and, AWG:22
installing, AR1:47-48
  new, AR1:146-147
summary of, AR1:33
instance data of, AWG:65
instances and, AWG:48
keys and, AR1:24
layout, AR1:381-398; PDT:16
learning about, AWG:118
linking, AR1:47
manager, AR1:423, 439-440
mask, AR2:415-416
message arguments for, AWG:54-55
messages, AWG:44
  Class Manager and, AWG:99
  defining set of, AWG:55
  overriding, AWG:44
  metaclasses, AR1:84-85
  method tables, AWG:55; AR1:119
  identifying, AWG:56
  MIL Service, AWG:273
  _NEW structure, AWG:49-51
  Notepaper App, AWG:265
  object
    confirming, AR1:54
    getting, AR1:55
  open service object, AR2:441
  organizing into program units,
  AWG:61
Paint, AWG:266
parts of, AR1:31-32
predefined, AR1:83
  with clsAppMgr, AR1:85
public, AWG:175
registering, AWG:175
return values and, AR1:14
root, PDT:14
service, AR2:255-264, 439-441
  installation, AR2:441
  service manager, AR2:440
sharing, AWG:174
simple, AR1:82-84
sources for, AWG:61
subclasuing, AWG:5
Template Application, AWG:251
Test Service, AWG:272-273
Text subsystem, AR2:7
  that respond to search messages,
  AR2:198
Tic-Tac-Toe, AWG:149-150, 205
  Toolkit Demo, AWG:268
UI Toolkit, AWG:18, 116-117;
  AR1:357-360
  inheriting from clsControl, AR1:359
  kinds of, AR1:361
  not inheriting from clsControl, AR1:358
  outline of, AR1:360
utility, PDT:19
Writerap, AWG:271
writing, that can be searched, AR2:196

see also specific classes and types of classes
Class implementation C files, AWG:109
CLASS INFO array, AR1:44-45
defined, AR1:42
entry fields, AR1:45
Class inheritance, PDT:14
Class initialization routine, AR1:47-48
Class Manager, AWG:7, 41-58;
PDT:13-14
capabilities and, AR1:25-29
class and object use, AWG:20
classes, AWG:43-45
creating, AWG:53-58
ClsSymbolsInit, AWG:162
code, AWG:42
  concepts, AR1:9-29
  constants, AWG:71-72
  conversion functions, AWG:163
  creating new objects and, AR1:15-18
  data structures, AWG:48-49
defined, PDT:13
document process and, AR1:90
  features supported by, PDT:14
  handling message status and,
  AR1:18-19
  identifiers, AR1:9-13
instance data, AWG:132, 140; AR1:34
  macros and, AWG:82
  memory protection and, AR1:34
  message handlers and, AWG:99, 109
  messages, AWG:42-43
  sending, AWG:45-48
  message-sending macros, AR1:23-24
  method entry points and, AR1:37
  method invocation, AR1:35
  method table, AWG:47, 49; AR1:41-42
  file, AWG:56
msgDestroy, AWG:109
msgDump, AWG:161
msgRestore, AWG:142
msgSend, AWG:84
msgSet, AWG:161
msgRestore, AWG:142
object
  creating, AWG:48-53
  pointer, AWG:47
observable objects and, AR1:78
overview, AR1:6
parameters for function, AWG:56
passing messages with, AR1:33
reapart process and, AR1:242-243
returned values and, AWG:110
scavenging and, AR1:60
sending messages, AR1:13-15
  other ways for, AR1:19-22
  symbolic names and, AWG:159
UID conversion, AWG:111
use of, AWG:76
using keys and, AR1:24-25
CLASS NEW ONLY structure, AR1:47-48, 146
contents, AR1:47-48
CLASS NEW structure, AR1:47

Clear command (MiniNote Edit menu),
  UI:136
ClExtend() macro, AR1:393
Client, AWG:42
clsCntr and, AWG:138-139
  in creating object, AWG:49
  defined, AWG:42
  in initializing _NEW structure, AWG:52
object communication, AWG:111
  window, AWG:120
  one, per frame, AWG:122
  see also Code
Client-defined attributes, AR2:54, 77-78
  transfer protocols, AR2:170
Client window, AR1:92
  closing document and, AR1:109
  creating, for application frame,
  AR1:212-213
  document termination and, AR1:112
  frame layout and, AR1:501
  positioning scroll window, AR1:461
Clipboard.
  see Windows clipboard
Clipping, AR1:219-221, 235
  children, AR1:221
  unclipped, AR1:236
  disadvantage of, AR1:220
  local, AR1:269, 270
  messages, AR1:283
  on image, AR1:219
  overriding, AR1:220
  rectangles, AR1:338
  region, AR1:219-220
  defined, AR1:235
  sharing parent's, AR1:221
  window, AR1:220
siblings, AR1:221
window, AR1:236
Clock, AWG:263-264; UI:13
  busy, UI:81
  icon, UI:77
Close box, AR1:507
Closed figures, AR1:271-272
  drawing, AR1:294
  messages, AR1:283-284
Closed icon, UI:74, 221-222
Closing
  address book, AR2:326
  application window, AR1:196
  documents, AR1:109-110
  files, AWG:145; AR2:46, 74-75
    sample code, AR2:46
    parallel port, AR2:277
    serial port, AR2:268
  service, AR2:262-263
  socket handle, AR2:299
  see also Opening
clsABMgr, messages, AR2:325
clsAddrBookApplication, AR2:320
clsClockApp, AWG:264
clsCloseBox, AR1:507
clsCnt, AWG:135
  getting and setting values,
  AWG:139–140
  instance data, AWG:138–139
  method table for, AWG:138
  object, AWG:136
clsCntApp, AWG:136
  instance data, AWG:142–143
  label and, AWG:136
  memory-mapped files and, AWG:143
  menu creation, AWG:146
  method table for, AWG:137–138
  msgSave, AR1:141
clsCodeInstallMgr, AR2:379, 414
  in installing applications or services,
  AR2:415
  messages, AR2:415
clsCommandBar, AR1:508; AR2:138
clsContainerApp, AR1:185, 186
clsControl, AR1:399
  button notification and, AR1:420
  control enable and, AR1:404
  dirty controls and, AR1:403
  in filing controls, AR1:399
  gesture handling and, AR1:369
  label notification and, AR1:414
  message dispatching and,
  AR1:399–400
  messages, AR1:401–402
  msgGWinGesture and, AR1:408
  previewing and, AR1:406–407
  subclasses values of, AR1:403
  UI Toolkit classes inheriting from,
  AR1:359
  UI Toolkit classes not inheriting
  from, AR1:358
  xgs1Tap gesture and, AR1:408
  see also Controls
clsCounter, AWG:136; AR1:509
  instance data and, AWG:139
clsCustomLayout, AWG:122; AR1:196, 353–354
  aligning edges and, AR1:393
  clsFrame and, AR1:389
  function, AR1:361, 381
  layout constraints, AR1:392
  specifying, AR1:390–391
  layout loop and, AR1:397
  layout of adjacent windows by,
  AR1:394
  messages, AR1:390
  shrink-wrap and, AR1:395
  window layout and, AR1:382
  see also Custom layout
clsDateField, AR1:486
clsDirHandle, AR2:58
  messages, AR2:64
clsDrwCtx, AR1:211
  storing graphic states and, AR1:277
clsEmbeddedWin, AWG:155; AR1:69,
  117; AR2:157
clsAppWin comparison, AR1:118
  in creating embedded window,
  AR1:190
  default behavior, AR1:118
  descendants of, AR1:118
  getting exact pen location and,
  AR1:123
  handles selection messages, AR2:161
  identifying selection and, AR1:120
  inheriting from, AR1:118
  message intercepts, AR1:127
  messages, AR1:118–119, 189
  in move and copy operations,
  AR1:191–192
  protocol, AR1:121
  move and copy, AR1:119
  Tic-Tac-Toe application and, AR1:123
  translations, AR1:119
  UI Toolkit and, AR1:370
  using, AR1:189–193
  see also Embedded window class
clsEmptyApp, AWG:99, 177
  message handling and, AWG:107
ClsEmptyAppInit, AWG:103
  code sample, AWG:108
clsExport, AR2:150
  messages, AR2:152
clsField, AWG:123; AR1:356, 475
  messages, AR1:476–477
clsFileHandle, AR2:375
  in creating file handle,
  AR2:348
  messages, AR2:64
clsFileStream, AR1:10; AR2:58, 124;
  PDT:19
  messages, AR2:62–63
clsFixedField, AR1:486
clsFontInstallMgr, AR1:441; AR2:378,
  380, 405
  functions, AR2:417
  instance of, AR2:406
  messages, AR2:417
clsFontListBox, AR1:463, 473
clsFoo, AWG:251
clsFrame, AWG:121; AR1:112
  creating close box and, AR1:507
  filing state and, AR1:503
  frame layout and, AR1:501
  messages, AR1:499
  modifying frame and, AR1:500
  page number creation and, AR1:509
  shrink-wrap and, AR1:395
tab bars and, AR1:508
  see also Frames
clsGotoButton, AR1:118
clsGrabBox, AR1:528
  messages, AR1:529
clsGWin, AWG:156, 269–270;
  AR1:368–370, 552, 617
clsScribble object, AR1:617
help gesture and, AWG:166; AR1:370
help ID field, AR1:617
message previewing and, AR1:407
messages, AR1:617–618
Quick Help, AR2:181–182
  messages and, AR2:187
  xgsQuestion gesture and, AR1:408
  see also Gesture windows
clsHelloWin, AWG:125; AR1:238
  classes used, AWG:187
DC creation, AWG:130, 131
DC state, AWG:131
drawing and, AWG:133
  enhancements, AWG:133
highlights, AWG:127–128
instance data, AR1:131–132
accessing, AWG:132
method table for, AR1:125, 127
painting and, AWG:129
structure definition, AWG:130
ClsHelloWinInit, AWG:130
clsHelloWorld, AWG:120, 125
  classes used, AWG:181, 187
  highlights, AWG:127
method table for, AR1:125, 127, 181
clsHWXInstallMgr, AR2:372
clsIcon
  bitmap picture and, AR1:524–525
  messages, AR1:523
  painting and, AR1:525
  pixmap picture and, AR1:525
  see also Icons
clsImgDev, AR1:210, 255
  creating imaging device and, AR1:256
  landscape and portrait mode and,
  AR1:262
  memory allocation and, AR1:258
clsImport, AR2:150
  messages, AR2:150
clsINBXService, AR2:305
  default behaviors, AR2:312
  I/O protocol, AR2:313
  messages, AR2:315
clsIniFileHandler, AR2:379
clsInputApp, AWG:270; AR1:561
clsInstallMgr, AR2:258, 375, 379
  advanced topics, AR2:414
  controlling items and, AR2:406–407
  installed item database and, AR2:406
instance of, AR2:376, 406
messages, AR2:405
class, AR2:409
instance, AR2:409–410
notification, AR2:408
subclass, AR2:410
using, AR2:409–413
observing installation mangers and,
AR2:407
semaphore use, AR2:414
observing installation managers and,
AR1:488
subclass, AR2:462
subclass of, AR2:405
clsIntegerField, AWG:136; AR1:486
clsInWin, AWG:269, 270; AR1:560
clsIOBXService, AR2:305
handling input/output and, AR2:312
messages, AR2:316
clsIP, AWG:269–270; AR1:552
clsPaper general facility, AR1:590
messages, AR1:585–586
using, AR1:585–588
creating insertion pad,
AR1:586–587
deleting insertion pad, AR1:587
displaying insertion pad, AR1:587
translator object, AR1:587–588
XList data, AR1:588
clsLabel, AWG:89; AR1:400
child windows and, AR1:415
Clock and, AWG:264
decoration drawing and, AR1:413
dirty controls and, AR1:403
field support and, AR1:416
function, AR1:409
Hello World (toolkit) and, AWG:117, 181
hierarchy, AWG:119
icon layout and, AR1:525
inherited classes from, AWG:122
messages, AR1:410
msgNew arguments for,
AWG:118–119
notification, AR1:414
painting and, AR1:415
style settings, AWG:119
see also Labels
clsList, AWG:45; AR1:83–84; AR2:127;
PDT:19
function, AR2:124
header file, AWG:45
messages, AR2:128
functions, AR2:127
method table, AWG:57–58
in UDH, AWG:53
clsListBox, AR1:463
entries and, AR1:465–466
gestures and, AR1:468
messages, AR1:449, 463–464
painting and, AR1:468–469
scrolling and, AR1:466
see also List boxes
clsManager, AR1:439
clsMark, AR1:69, 129; AR2:196
embedded window marks and,
AR1:118
instance (mark), AR1:129
link messages, AR1:134
messages
class, AR1:199
holders send to marks, AR1:199
marks sent to components,
AR1:199–200
messages sent to components,
AR1:200
sent internally, AR1:200
protocol, AR1:133
see also Marks class
clsMenu, AR1:5, 445
creating menus and, AR1:447–448
displaying menu and, AR1:448
messages, AR1:447
see also Menus
clsMenuButton, AR1:363, 445
messages, AR1:445
clsAMGR.H, AWG:109; PDT:128
clsMILAsyncSIODevice, AR2:265
concurrency and, AR2:266
messages, AR2:267
structures, AR2:265
clsMilSvc, AR2:449
clsModalFilter, AR1:488
clsModern, AR2:279
API, AR2:281
bypassing, AR2:290
commands for establishing
connection, AR2:287
creating, object, AR2:282–283
messages, AR2:281–282
waiting for connection and, AR2:289
clsMoveCopyIcon, AR1:125
clsMyView, AR1:517–519
clsNBAp, AR1:186–187
clsNote, AWG:168; AR1:488
argument structure, AR1:15–16
creating new instance of, AR1:18
filter, AR1:490–491
label creation, AR1:489
message handling status and, AR1:18
messages, AR1:488
painting and, AR1:492
StdMsg customization, AWG:171
see also Notes
clsNoteIconWin, AWG:264
clsNotePaper, AR2:124, 229; PDT:19
coordinate system, AR2:229
messages, AR2:231
metrics, AR2:230
view, AR2:229
clsNotePaperApp, AWG:265
clsNPData, AR2:124, 229
messages, AR2:233–234
note paper data and, AR2:232
clsNPItem, AR2:124, 229
instances, AR2:234
messages, AR2:234–235
note paper data and, AR2:232
clsNPScrubbleItem, AR2:234
clsNPTextItem, AR2:234
ClsNum, AWG:156
clsObject, AWG:51; AR1:5–6
application classes and, AR1:87
class and, AR1:82
clsView and, AR1:174
data structure and, AWG:49
instance of, AWG:28
method tables and, AR1:42
msgCopy and, AR1:49–50
msgFree and, AR1:59
msgFreeOK and, AR1:58
object/class information messages and,
AR1:54
saving/restoring data and, AR1:77
saving/restoring instance data and,
AR1:35
in system process, AR1:82–84
UID and, AWG:52
see also Objects
clsObjectToString, AWG:163
clsOBJXService, AR2:305
default behaviors, AR2:310
existing Out box services and, AR2:311
messages, AR2:310, 314
Out box document response to,
AR2:310
writing own Out box service and,
AR2:310
clsOpenServiceObject, AR2:441, 450, 452
clsService and, AR2:471
function, AR2:471
subclassing, AR2:471–472
clsOption, AR1:508, 511
card destruction and, AR1:516
command sheets and, AR1:522
indicating mixed attributes and,
AR1:520
messages, AR1:512–513
clsOptionTable, AR1:521
clsPageNum, AWG:136; AR1:509
clsParallelPort, AR2:275
messages, AR2:276
structures, AR2:275
clsPicSeg, AR1:270, 279, 317
graphics and, AR1:319–320
graphics applications and,
AR1:323–326
hit testing and, AR1:325
messages, AR1:317–318
drawing, AR1:319
seen also

clsPixDev, AR1:210, 266
clsPopupChoice, AR1:450

messages, AR1:450, 451
clsPreferences, AR2:361
classPrint, AR1:140
classPrLayout, AR1:139
classProgressBar, AR1:531

messages, AR1:535
	inherited, AR1:540

text metrics, AR1:532

see also

Progress bar

clsQuickHelp, AR2:182

messages, AR2:187

using, AR2:187-188
classResFile, AWG:145; AR1:70; AR2:67

messages, AR2:347-348
	msgNew and, AR1:111, 113
	saving and restoring data, AR1:77
	using, AR2:347-354
classResList, AR2:345

in creating resource list, AR2:346
classRootContainerApp, AR1:185, 186

concepts, AR1:187

messages, AR1:187
classScribble, AR1:552, 607

messages, AR1:608
classScrollbar, AR1:453
classScrollWin, AR1:457

creating scrollwin and, AR1:458

messages, AR1:458
	option card layout and, AR1:515

scrollwin layout and, AR1:460-461
classSectorApp, AWG:34
classSelectionMgr, AR1:439

messages, AR1:440
classSelection, AR1:128; AR2:155

instance, AR2:155

message categories, AR2:156

messages, AR2:157-158
	from clients to
		theSelectionManager, AR2:158-159
classSendableService, AR2:319, 331
classService, AR2:255

handling msgNewDefaults, AR2:457

handling of msgNew, AR2:457

messages

change ownership protocol, AR2:467-469

information, AR2:459

notification, AR2:461-462

responsibility, AR2:469-470

msgSvcClassLoadInstance and, AR2:459

msgSvcOpenRequested and, AR2:463

object-oriented architecture and,

AR2:449

service instances and, AR2:439-440

service manager messages and,

AR2:459

classServiceInstallMgr, AR2:405, 416, 441

instance of, AR2:406

messages, AR2:416

classServiceMgr, AR2:258, 260

concepts, AR2:440

messages, AR2:260

opening and closing service and,

AR2:262

classShadow, AR1:509

classSio, AR2:124, 280; PDT:19

classSPaper, AR1:552, 589-596

creating subclass, AR1:593

instance of, AR1:593-594

facilities, AR1:590-592

functions, AR1:589

input flags, AR1:594

messages, AR1:589-590

parsing XList data, AR1:592

rendering translated text, AR1:592

subclass, AR1:592-593

subclass instance, AR1:593-594

translator, AR1:595-596

classScrollWin, AR1:243

classStream, AR2:133; PDT:19

function, AR2:124

messages, AR2:133

services and, AR2:470

writing agents and, AR2:354

stream transfers and, AR2:168

subclassing, AR2:133

subclass instance, AR2:124

classSysDc, AR1:285

classSysDrwCtx, AR1:210-211, 235, 267

colors and, AR1:295

in coordinate system transformations, AR1:270

drawing context features and, AR1:270

drawing context and, AR1:270

drawing and, AR1:270

hit detection and, AR1:270

messages, AR1:281-284

drawing, AR1:319

picture segments and, AR1:319

raster operation and, AR1:292-293

toolkit table buttons and, AR1:425

window layout and, AR1:382

see also

Table layout
classTableApp, AWG:251
classTestOpenObject, AR2:450
classTestOpenObjectInit routine, AR2:451, 452

classTestServiceInit routine, AR2:451
classTestService method table, AR2:457

classTestSvc, AR2:449
classText, AR1:105; AR2:198

embedded objects and, AR2:20

messages, AR2:12-13

for changing attributes, AR2:7

observer, AR2:21

Text subsystem and, AR2:3

classTexteditApp, AR2:180

classTextField, AR1:486

classTextIP, AR2:33

messages, AR2:33

classTextView, AR1:517-519

creating object of, AR2:26

defined, AR2:9

in insertion pad creation, AR2:26

messages, AR2:23-24

msgNewDefaults for, AR2:24-25

classTiff, AR1:331-333

messages, AR1:331

TIFF object creation and, AR1:332

toolkit table buttons and, AR1:425

window layout and, AR1:382

see also

Table layout
classTemplateApp, AWG:159, 162

classSysDc, AR1:285

classSysDrwCtx, AR1:210-211, 235, 267

colors and, AR1:295

in coordinate system transformations, AR1:270

drawing context features and, AR1:270

drawing context and, AR1:270

drawing and, AR1:270

hit detection and, AR1:270

messages, AR1:281-284

drawing, AR1:319

picture segments and, AR1:319

raster operation and, AR1:292-293

sampled images and, AR1:272

in sending window messages, AR1:289

see also

Drawing context (DC)
classSystem, AR2:429

messages, AR2:431

paths for file system constants,

AR2:430
classTabBar, AR1:223, 508

subclasses of clsTkTable and, AR1:442
classTabButton, AR1:220, 221
classTable, AR2:213; PDT:19

creating table object and, AR2:216
data files and, AR2:214

function, AR2:124

library support routines, AR2:213

messages, AR2:217-218

information, AR2:226

requesting new position from,

AR2:215

semaphore and, AR2:217
classTabLayout, AWG:122; AR1:353, 381

Calc's positioning of child window

using, AR1:388

control enable and, AR1:450

function, AR1:361, 383

layout loop and, AR1:397

messages, AR1:384
table window constraints,

AR1:386-387

toolkit table buttons and, AR1:425

window layout and, AR1:382

see also

Table layout
classTestOpenObject, AR2:450
classTestOpenObjectInit routine, AR2:451, 452

classTestServiceInit routine, AR2:451
classTestService method table, AR2:457

classTestSvc, AR2:449
classText, AR1:105; AR2:198

eMBEDDED objects and, AR2:20

messages, AR2:12-13

for changing attributes, AR2:7

observer, AR2:21

Text subsystem and, AR2:3

classTexteditApp, AR2:180

classTextField, AR1:486

classTextIP, AR2:33

messages, AR2:33

classTextView, AR1:517-519

creating object of, AR2:25

defined, AR2:9

in insertion pad creation, AR2:26

messages, AR2:23-24

msgNewDefaults for, AR2:24-25

classTiff, AR1:331-333

messages, AR1:331

TIFF object creation and, AR1:332
linker command files, AWG:67
method table files, AWG:66
SDK files, AWG:67
WATCOM, AWG:66
Completion note, UI:211
timing-triggered, UI:212–213
Component, AWG:20, 40
application, AWG:149–150
classes, AWG:66; AR:176
saving/restoring data and, AR:177
layer, AWG:12
defined, AWG:6
Notebook, AWG:29
objects, AR:192
Component (mark), AR:130
Component, AWG:20, 40
Completion note,
volume protection, AR:67
VI, AR:349
validating, AR:201
serial file location, AR:67
for modular design,
frame, AR:497
field, creation, AR:480–481
decoration window,
application, AR:76–77, 349
ancestor, AR:201
illustrated, UI:18
laying out, AR:349
for modular design, AWG:15
nested, AR:362–363
how menus work and, AR:363
option sheets, AR:1521
UI, AR:1349
UI Toolkit, AR:1350
creating, AWG:116–119
filed representation and,
AR:364–365
filling, AR:365
illustrated, AWG:117
nested, AR:362–363
ComposeText functions, AWG:64; AR:2114
Compound documents, UI:18
illustrated, UI:18
Concurrency considerations, AR:266–67
file location, AR:267
protecting file data, AR:266–67
serial I/O, AR:266
volume protection, AR:267
Config keyword, PDT:37
DebugTablet, PDT:37, 40–41
Configurations, PDT:37
data modem, AR:283–287
machine, PDT:26
modification, UI:164
monitors and, PDT:44
options, UI:165
parallel port, AR:227
serial port, AR:268–271
data modem, AR:280
setting up specific, PDT:44–45
tablet-like, PDT:40–42
user, of application, UI:224
Configuring
digitizing tablet, PDT:45
mouse, PDT:45
Confirmation notes, UI:212
Connected Disks page (Connections notebook), UI:106–113
clash notes and, UI:71
Create Directory command, UI:195
disk representation in, UI:73
edit pads and, UI:47
icons and, UI:106
menus, UI:107
option sheets, UI:111–112
icon, UI:75
options menu, UI:111
Quick Installer, UI:113
controls, UI:112
views, UI:108–110, 113
Applications view, UI:110
Bookshelf view, UI:109
Directory view, UI:108
Fonts view, UI:110
installable software view,
UI:109–110
Connected Printers page (Connections notebook), UI:116–118
illustrated, UI:116
menus, UI:116–117
options sheets, UI:117–118
Connecting, volumes, AR:250
Connection
service, AR:2258, 446–447
socket, AR:296
Connectionless communication, AR:2296
see also Datagram, delivery
Connection-oriented communication,
AR:2296
Connections icon, UI:13, 76
Connections notebook, AWG:8;
UI:105–119; AR:196; AR:2250
Attached Printers page, UI:163
Connected Disks page, UI:106–113
clash notes and, UI:71
Create Directory command, UI:195
disk representation in, UI:73
edit pads and, UI:47
icon option sheet and, UI:75
views, UI:108–110
Connected Printers page, UI:57, 116–118
disks page, PDT:51
for displaying application, AWG:67
for importing bitmap, PDT:168
installing services through, AR:2256
Network View page (Disks section),
UI:114–115
Network View page (Printers section),
UI:118–119
overview, UI:105
quick installation and, AR:2397
socket instance and, AR:298
table of contents, UI:105
see also Auxiliary notebooks; Help
notebook; Notebook;
Settings notebook;
Stationary notebook
Connectivity, AR:2245–250; PDT:20
adding network protocols and,
AR:2251
additional information on, AR:2242
computer, AR:2244–245
facilities, AR:2250
introduction to, AR:2241
MIL services, AR:2245–246
other services and, AR:2246–249
principles of, AR:2243
remote interfaces and, AR:2251–253
service manager and, AR:2250
services and interfaces, AR:2249–250
strategies, AR:2244
Consistency, UI:154
departure from standards and, UI:227
toggle gesture, UI:247
in UI design, UI:149
CONSOLE.DLC file, PDT:43, 137
description, PDT:28
Constants, AWG:71–72
basic, AWG:77
Constraining translation, UI:241–243
ambiguity and, UI:241–242
with dictionary and templates,
UI:242–243
to letters, numbers, punctuation,
symbols, UI:242
Constraints, UI:241
context-specific, UI:242
custom layout, AR:1392
alignment, AR:1393
flags, AR:1395
four child window, AR:1391
specifying, AR:1390–391
types of, AR:1392
guidelines for using, UI:243
shrink-wrap and, AR:1395–396
relative window, AR:1396
value, AR:1396
table layout, AR:1386–387
Container application classes,
AR:185–188
contents page and, AR:188
hierarchy, AR:186
reference document and, AR:188
root, AR:186–188
see also Embedded document
Containers, UI:40
Context, PDT:73
inside breakpoints, PDT:131
Control buttons, UI:28, 175–180
<table>
<thead>
<tr>
<th>Topic</th>
<th>Page(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Counter Application, AWG:135–140 classes, AWG:196</td>
<td>238</td>
</tr>
<tr>
<td>creating, AWG:137, 196</td>
<td></td>
</tr>
<tr>
<td>counter class and, AWG:138 files, AWG:196</td>
<td></td>
</tr>
<tr>
<td>getting and setting values, AWG:139–140</td>
<td></td>
</tr>
<tr>
<td>highlights, AWG:137–138</td>
<td></td>
</tr>
<tr>
<td>installing, AWG:137</td>
<td></td>
</tr>
<tr>
<td>instance data, AWG:138–139</td>
<td></td>
</tr>
<tr>
<td>objectives, AWG:195</td>
<td></td>
</tr>
<tr>
<td>running, AWG:196</td>
<td></td>
</tr>
<tr>
<td>sample code, AWG:195–204</td>
<td></td>
</tr>
<tr>
<td>tutorial, AWG:89</td>
<td></td>
</tr>
<tr>
<td>Counter object, AWG:89</td>
<td></td>
</tr>
<tr>
<td>creating, AWG:143–144</td>
<td></td>
</tr>
<tr>
<td>filing, AWG:145–146</td>
<td></td>
</tr>
<tr>
<td>restoring, AWG:146</td>
<td></td>
</tr>
<tr>
<td>saving, AWG:145–146</td>
<td></td>
</tr>
<tr>
<td>Counting changes, AR2:37</td>
<td></td>
</tr>
<tr>
<td>list items, AR2:130</td>
<td></td>
</tr>
<tr>
<td>CPU, power conservation and, AWG:7</td>
<td></td>
</tr>
<tr>
<td>Create command (Edit menu), UI:195</td>
<td></td>
</tr>
<tr>
<td>createDataObject argument, AR1:174</td>
<td></td>
</tr>
<tr>
<td>createInitial style bit, AR2:406</td>
<td></td>
</tr>
<tr>
<td>Create menu() function, AR1:556</td>
<td></td>
</tr>
<tr>
<td>Create menu, UI:20</td>
<td></td>
</tr>
<tr>
<td>caret gesture family and, UI:233</td>
<td></td>
</tr>
<tr>
<td>Connected Printers page, UI:117</td>
<td></td>
</tr>
<tr>
<td>system-wide, UI:120</td>
<td></td>
</tr>
<tr>
<td>table of contents, UI:86–87</td>
<td></td>
</tr>
<tr>
<td>Creating</td>
<td></td>
</tr>
<tr>
<td>address descriptors, AR2:331–332</td>
<td></td>
</tr>
<tr>
<td>address windows, AR2:333–334</td>
<td></td>
</tr>
<tr>
<td>application directory handle, AR1:170</td>
<td></td>
</tr>
<tr>
<td>application distribution volume, AR2:391</td>
<td></td>
</tr>
<tr>
<td>application main window, AR1:504</td>
<td></td>
</tr>
<tr>
<td>Auxiliary notebook</td>
<td></td>
</tr>
<tr>
<td>documents, AR2:425</td>
<td></td>
</tr>
<tr>
<td>sections, AR2:424</td>
<td></td>
</tr>
<tr>
<td>bitmaps, AR1:330; PDT:192–193</td>
<td></td>
</tr>
<tr>
<td>border windows, AR1:373–375</td>
<td></td>
</tr>
<tr>
<td>browser, AR2:138</td>
<td></td>
</tr>
<tr>
<td>object, AR2:140</td>
<td></td>
</tr>
<tr>
<td>buttons, AR1:354, 419–420, 421</td>
<td></td>
</tr>
<tr>
<td>many, AR1:422</td>
<td></td>
</tr>
<tr>
<td>byte buffer object, AR2:208</td>
<td></td>
</tr>
<tr>
<td>cached images, AR1:300</td>
<td></td>
</tr>
<tr>
<td>choice, AR1:352–353, 443</td>
<td></td>
</tr>
<tr>
<td>classes, AR1:6, 31–48</td>
<td></td>
</tr>
<tr>
<td>client window, AR1:212–213</td>
<td></td>
</tr>
<tr>
<td>close box, AR1:507</td>
<td></td>
</tr>
<tr>
<td>clsModem object, AR2:282–283</td>
<td></td>
</tr>
<tr>
<td>clsSPaper subclass, AR1:593</td>
<td></td>
</tr>
<tr>
<td>instance of, AR1:593–594</td>
<td></td>
</tr>
<tr>
<td>command sheets, AR1:522</td>
<td></td>
</tr>
<tr>
<td>control, AR1:402–403</td>
<td></td>
</tr>
<tr>
<td>custom layout, AR1:353–354</td>
<td></td>
</tr>
<tr>
<td>window, AR1:390–395</td>
<td></td>
</tr>
<tr>
<td>DC, AR1:285</td>
<td></td>
</tr>
<tr>
<td>directories, AR2:69–74</td>
<td></td>
</tr>
<tr>
<td>browser and, AR2:141</td>
<td></td>
</tr>
<tr>
<td>indexes, AR2:80</td>
<td></td>
</tr>
<tr>
<td>directory handles, AR2:58, 60, 71</td>
<td></td>
</tr>
<tr>
<td>DLLMain, AR2:450</td>
<td></td>
</tr>
<tr>
<td>documents, AR1:102, 148</td>
<td></td>
</tr>
<tr>
<td>document with browser, AR2:141</td>
<td></td>
</tr>
<tr>
<td>embedded application, AR1:196</td>
<td></td>
</tr>
<tr>
<td>embedded documents, UI:20</td>
<td></td>
</tr>
<tr>
<td>embedded window, AR1:190</td>
<td></td>
</tr>
<tr>
<td>field, AR1:477–479</td>
<td></td>
</tr>
<tr>
<td>file handles, AR2:71–73</td>
<td></td>
</tr>
<tr>
<td>files, AR2:69–74</td>
<td></td>
</tr>
<tr>
<td>font list boxes, AR1:473</td>
<td></td>
</tr>
<tr>
<td>frames, AR1:500</td>
<td></td>
</tr>
<tr>
<td>handles, AR2:69–70</td>
<td></td>
</tr>
<tr>
<td>header file, AR1:36</td>
<td></td>
</tr>
<tr>
<td>help text, AR2:180</td>
<td></td>
</tr>
<tr>
<td>hints, PDT:192</td>
<td></td>
</tr>
<tr>
<td>icons, AR1:524–525</td>
<td></td>
</tr>
<tr>
<td>imaging device, AR1:256</td>
<td></td>
</tr>
<tr>
<td>insertion pads, AR1:586–587</td>
<td></td>
</tr>
<tr>
<td>installable-item managers, AR2:410</td>
<td></td>
</tr>
<tr>
<td>IP window, AR1:556–557</td>
<td></td>
</tr>
<tr>
<td>lists, AR2:129</td>
<td></td>
</tr>
<tr>
<td>marks, AR1:133; AR2:196</td>
<td></td>
</tr>
<tr>
<td>menu bar, AR1:353</td>
<td></td>
</tr>
<tr>
<td>menu buttons, AR1:446–447</td>
<td></td>
</tr>
<tr>
<td>menus, AR1:447–448</td>
<td></td>
</tr>
<tr>
<td>methods, AR1:37–41</td>
<td></td>
</tr>
<tr>
<td>method tables, AR1:41–46</td>
<td></td>
</tr>
<tr>
<td>nodes, AR2:43</td>
<td></td>
</tr>
<tr>
<td>notes, AR1:488–490</td>
<td></td>
</tr>
<tr>
<td>objects, AR1:15–18, 34</td>
<td></td>
</tr>
<tr>
<td>with default values, AR1:18</td>
<td></td>
</tr>
<tr>
<td>option sheets, AR1:514</td>
<td></td>
</tr>
<tr>
<td>picture segment, AR1:318–319, 322</td>
<td></td>
</tr>
<tr>
<td>pop-up choices, AR1:451</td>
<td></td>
</tr>
<tr>
<td>progress bar, AR1:535</td>
<td></td>
</tr>
<tr>
<td>Quick Install disk, AR2:397</td>
<td></td>
</tr>
<tr>
<td>receiver’s stream, AR2:176–177</td>
<td></td>
</tr>
<tr>
<td>resource file handle, AR2:348–349</td>
<td></td>
</tr>
<tr>
<td>resource lists, AR2:346</td>
<td></td>
</tr>
<tr>
<td>scribbles object, AR1:609</td>
<td></td>
</tr>
<tr>
<td>scrollwin, AR1:458–459</td>
<td></td>
</tr>
<tr>
<td>sender’s stream, AR2:177</td>
<td></td>
</tr>
<tr>
<td>service instances, AR2:442, 454–455</td>
<td></td>
</tr>
<tr>
<td>shadows, AR1:509</td>
<td></td>
</tr>
<tr>
<td>stream objects, AR2:134</td>
<td></td>
</tr>
<tr>
<td>string list boxes, AR1:470–471</td>
<td></td>
</tr>
<tr>
<td>string object, AR2:212</td>
<td></td>
</tr>
<tr>
<td>submenus, AR1:448</td>
<td></td>
</tr>
<tr>
<td>tab bars, AR1:508</td>
<td></td>
</tr>
<tr>
<td>table object, AR2:220</td>
<td></td>
</tr>
<tr>
<td>tables, AR2:221</td>
<td></td>
</tr>
<tr>
<td>tabular window layout, AR1:353</td>
<td></td>
</tr>
<tr>
<td>temporary files and, AR2:65</td>
<td></td>
</tr>
<tr>
<td>text data object, AR2:7, 13</td>
<td></td>
</tr>
<tr>
<td>text insertion pads, AR2:33</td>
<td></td>
</tr>
<tr>
<td>text views, AR2:9, 24–26</td>
<td></td>
</tr>
<tr>
<td>TIFF object, AR1:332</td>
<td></td>
</tr>
<tr>
<td>title bar, AR1:507</td>
<td></td>
</tr>
<tr>
<td>tokens, AR1:200</td>
<td></td>
</tr>
<tr>
<td>toolkit table, AR1:428–435</td>
<td></td>
</tr>
<tr>
<td>child windows, AR1:434</td>
<td></td>
</tr>
<tr>
<td>translator, AR1:605</td>
<td></td>
</tr>
<tr>
<td>window, AR1:212</td>
<td></td>
</tr>
<tr>
<td>size and position, AR1:232</td>
<td></td>
</tr>
<tr>
<td>style flags, AR1:232</td>
<td></td>
</tr>
<tr>
<td>XList, AR1:614</td>
<td></td>
</tr>
<tr>
<td>see also Deletions Removing</td>
<td></td>
</tr>
<tr>
<td>Creation capability, AR1:28</td>
<td></td>
</tr>
<tr>
<td>Creation notification, AR1:28</td>
<td></td>
</tr>
<tr>
<td>Cross Out gesture, UI:16, 24</td>
<td></td>
</tr>
<tr>
<td>default targets, UI:232</td>
<td></td>
</tr>
<tr>
<td>in gesture mode, UI:258</td>
<td></td>
</tr>
<tr>
<td>guidelines for, UI:237</td>
<td></td>
</tr>
<tr>
<td>hot point for, UI:231</td>
<td></td>
</tr>
<tr>
<td>MiniNote gesture mode, UI:135, 141</td>
<td></td>
</tr>
<tr>
<td>MiniNote ink mode, UI:135</td>
<td></td>
</tr>
<tr>
<td>tab stops and, UI:132</td>
<td></td>
</tr>
<tr>
<td>see also Gestures</td>
<td></td>
</tr>
<tr>
<td>C run-time library, AR2:109–114</td>
<td></td>
</tr>
<tr>
<td>16-bit character support, AR2:110–114</td>
<td></td>
</tr>
<tr>
<td>ANSI standard C routines, AR2:109</td>
<td></td>
</tr>
<tr>
<td>files, AR2:109</td>
<td></td>
</tr>
<tr>
<td>time and date preferences, AR2:110</td>
<td></td>
</tr>
<tr>
<td>C run-time routines, PDT:128</td>
<td></td>
</tr>
<tr>
<td>CSTM_LAYOUT_CHILD_SPEC structure, AR1:391</td>
<td></td>
</tr>
<tr>
<td>CSTM_LAYOUT_DIMENSION structure, AR1:391–392</td>
<td></td>
</tr>
<tr>
<td>CstmLayoutSpecInit, AR1:396</td>
<td></td>
</tr>
<tr>
<td>cctx command, PDT:73</td>
<td></td>
</tr>
<tr>
<td>datasheet, PDT:94–95</td>
<td></td>
</tr>
<tr>
<td>frame numbers and, PDT:75</td>
<td></td>
</tr>
<tr>
<td>scopeSpec and, PDT:87</td>
<td></td>
</tr>
<tr>
<td>cctx parameter, AWG:99, 109; AR1:35 defined, AR1:38–39</td>
<td></td>
</tr>
<tr>
<td>ObjectCallAncestor() and, AR1:36</td>
<td></td>
</tr>
<tr>
<td>CTYP.E.H, AR2:112</td>
<td></td>
</tr>
<tr>
<td>Current selection, AR2:155</td>
<td></td>
</tr>
<tr>
<td>getting, text view, AR2:30–31</td>
<td></td>
</tr>
<tr>
<td>Writing Paper application, AR2:31</td>
<td></td>
</tr>
<tr>
<td>Cursor, AWG:4</td>
<td></td>
</tr>
<tr>
<td>I-beam, UI:283</td>
<td></td>
</tr>
<tr>
<td>Customize button (Installed Handwriting page), UI:100</td>
<td></td>
</tr>
<tr>
<td>Customizing buttons, UI:177–178</td>
<td></td>
</tr>
<tr>
<td>edit pads, UI:189</td>
<td></td>
</tr>
<tr>
<td>option sheets, UI:201–202</td>
<td></td>
</tr>
<tr>
<td>Custom layout, AR1:389–390</td>
<td></td>
</tr>
<tr>
<td>creating, AR1:353–354, 390–395</td>
<td></td>
</tr>
</tbody>
</table>
Data
application global, AR2:394–395
buffered, AR2:265
conversion/checking, AWG:78
duplication, AWG:15–16
capsulation, AWG:42
formats, AWG:17
input, AWG:4
interaction and view, AWG:152–153
reading, AR2:45
resource, AR2:337, 342, 353
C language definition, AR2:355
reading, AR2:349
writing and updating,
AR2:349–350
saving and restoring, AWG:135–148
sending and receiving via modem,
AR2:289
service, storage, AR2:455
structures, designing, UI:166
table
files, AR2:214
getting, AR2:223–224
setting, AR2:223
transaction, AR2:201
transfer, AWG:12
design checklist and, UI:296
refusing, UI:289
transfer type, AR2:166
tags, AR2:166–167
types, AWG:76–77
writing, AR2:44–45
see also Data modem, interface; Text
data object
dataAddress, PDT:88
Database
installed item, AR2:406
using tables in, AR2:217
Datagram
delivery, AR2:296
transaction services, AR2:296
types of, AR2:296
receiving, AR2:300
sending, AR2:299
Data modem
AT command set, AR2:290–293
characteristics, AR2:284
configuring, AR2:283–287
auto-answer mode, AR2:284
carrier state, AR2:284
command and data modes,
AR2:285
dial type, AR2:284
duplex mode, AR2:286
MNP mode, AR2:286
sending own AT commands,
AR2:286–287
speaker, AR2:284–285
connection types, AR2:285
direct communication with,
AR2:290–293
establishing connection with, AR2:287
interface, AR2:279–293
clsModemAPI, AR2:281
clsModem messages, AR2:281–290
concepts, AR2:279–280
configuration, AR2:280
direct communication with,
AR2:290–293
MNP data communication and,
AR2:289–290
reset settings, AR2:283
sending and receiving data with,
AR2:289
waiting for connection with, AR2:289
Data mode, modem, AR2:285
Data object
clsView and, AWG:27–28
design, AWG:152–153
dumping, AWG:161
saving, AWG:153
separate stateful, AWG:150
view and, AWG:155
Datasheets, AR2:95
Datasheets, command, PDT:89–112
date format preference, AR2:367
date preferences (Settings notebook),
UI:94
Date/time services
alarm services, AR2:103
current time, AR2:104
object-oriented timer interface,
AR2:104–105
timer routines, AR2:103
see also Time
DbgFlagGet(), AWG:86, 160; PDT:135
DbgFlagSet(), AWG:86; PDT:135
Dbg macro, PDT:133
DB.INI file, PDT:80–81
DBk task, PDT:152
Dbi task, PDT:151
DBm task, PDT:152
DB. see Source level debugger (DB)
DC. see Drawing context (DC)
d, db, dw, dd, commands, PDT:96
DDE (Dynamic Data Exchange) linking,
AWG:12
Deactivating
fields, AR1:481–482
hiding controls vs., UI:209
Debug
flag, PDT:38
set, PDT:128, 147
output port, PDT:29
stream data, PDT:37
tools, PDT:67
Window accessory, PDT:38
DEBUG compiler option, AWG:81, 87;
PDT:133–134
assertions and, AWG:85
debg flags and, AWG:85
defined, PDT:133
dumping and, AWG:161
PenPoint uses, PDT:133–134
preprocessor variable, AWG:82
tracing and, AWG:159–160
using, PDT:134
versions of DLLs, PDT:134
Debug() function, AWG:68, 84, 111;
PDT:67
debg flags and, AWG:160–161
Tic-Tac-Toe and, AWG:159
Debug flags, AWG:32, 85–86
B, AWG:96
D, AWG:112
Debug statements and, AWG:160–161
F1, AWG:97
F, AWG:97
G, AWG:97
sets, AWG:160
Debugger
flags
function of, PDT:133
setting, PDT:38
preparing to run, PDT:69–70
compiling and linking, PDT:69–70
files used in DB session, PDT:69
installing applications to debug,
PDT:70
installing DB, PDT:70
starting PenPoint, PDT:70
stream, PDT:37, 135–139
buffer, PDT:37
configuring destinations,
PDT:135–137
defined, PDT:133
different ways to view info sent to,
PDT:135
system log application and,
  PDT:141–143
writing to, PDT:137–139
see also Mini-debugger; Source level
debugger (DB)
Debugger stream, AWG:85, 111–112
using output, AWG:111
viewing output, AWG:111–112
Debugger() system routine, PDT:145
Debugging, AWG:68, 159–163
application behavior, PDT:43
assistance, AWG:84–87
assertions, AWG:85
debug flags, AWG:85–86
printing debugging strings,
AWG:84–85
suggestions, AWG:87
flag sets, AWG:86; PDT:134–135
clearing, PDT:148
setting, AWG:86
setting values in, PDT:135
general, techniques, PDT:133–139
DEBUG compiler option, PDT:133
debugger stream, PDT:135–139
debugging flag sets, PDT:134–135
Hello World (custom window),
AWG:133–134
Intel assembly language and, PDT:7
macros, PDT:67
messages, AWG:87
Penpoint 2.0 and, AWG:63
source code, PDT:72–73
finding and loading symbols,
PDT:72
using source code, PDT:72–73
strings, printing, AWG:84–85
symbolic information, PDT:72
loading partial, PDT:82
testing return values and,
AWG:115–116
Tic-Tac-Toe, AWG:159
windows, AR1:252
DebugLogFlushCount keyword, PDT:37
DebugLog keyword, PDT:37
DebugSet keyword, PDT:38
DEBUG warning macro, AR1:222
Decoration window components,
AR1:497
Default attributes
changing, AR2:8
text data objects, AR2:7
Default document
icon, UI:77
option sheets, UI:44
Default menus, UI:56
Document menu, UI:57–61
Edit menu, UI:62–65
Options menu, UI:44, 66–67
Default objects, UI:46
determining, UI:203
Default PenPoint application, UI:17
Default values, AR1:17
changing, AR1:18
#define, AWG:74
for constants, AWG:76
name, AWG:50
NewFields, AR1:16
Defines, AWG:71
file section, AWG:74
Definition file, method table, AR1:41
creating, AR1:42–45
Deleting
address book entry, AR2:328
Auxiliary notebook section/document,
AR2:426
bitmaps, PDT:193
characters, PDT:182
directories, AR2:75
with browser, AR2:141
forcing, AR2:75–76
documents, AR1:115
files, AR2:75
with browser, AR2:141
forcing, AR2:75–76
hints, PDT:192
insertion pads, AR1:587
many characters, AR2:11
resources, AR2:352–353
segments, PDT:186
shapes, PDT:188
tabular rows, AR2:224
tokens, AR1:201
XLIst element, AR1:614
see also Removing
Descendant classes, AWG:25
of clsApp, AWG:26
inheritance and, AWG:43
for storing structured data, AWG:28
Deselecting, UI:279
an existing selection, UI:280
Design
checklist, UI:295–296
graphic, UI:149
guidelines, AWG:15–18
modular, UI:152–153
user-centered, UI:149
wording in, UI:150
Designing
application icons, UI:216–222
guidelines for, UI:219–221
masks for, UI:221–222
applications, AWG:59–61
classes, AWG:60
consistency and, UI:154
data structures, UI:166
editing pads, UI:189
for Embedded Document
Architecture, UI:152–153
handwriting translators, UI:241–243
for internationalization and
localization, AWG:61–64
menu line, UI:226
message handlers, AWG:60
messages, AWG:60
for notebook metaphor, UI:151
for the pen, UI:152
for pen-based mobile computing,
UI:151–154
program units, AWG:61
for scalability, UI:153–154
user interface, AWG:60
user models and, UI:245–246
Desktop, AWG:33
floating accessories and, AWG:34
parent window, AWG:33
Destination application, AWG:11
Destroying
application directory handle, AR1:179
embedded window, AR1:190
image device, AR1:262
lists, AR2:131
notes, AR1:492
option card, AR1:516
string list boxes, AR1:472
text insertion pads, AR2:33
TIFF file, AR1:333
trackers, AR1:528
window, AR1:218
devCode, AR1:563
defined, AR1:569
in INPUT_EVENT structure, AR1:567
Development
application, AWG:59–90; PDT:6–7
checklist for, AWG:68–69
cycles, AWG:66–68
application installation, AWG:67
debugging and linking, AWG:66–67
general, AWG:88
debugging, AWG:68
iterative, UI:150
key concepts for, AWG:115
milestones, UI:5
options, PDT:5–7
strategy, AWG:64–66
component classes and, AWG:66
displaying on screen and, AWG:66
time point, AWG:65
instance data, AWG:65
stateful objects and, AWG:65
Device
applications, drivers and, AR2:246
Device List (Show menu),
device-independent RGB method,
Dial type, modem, AR2:284
Dialog sheets,
Dialogs, system and application, AR1:493
Dialog sheet, UI:112
Dialog string modifiers, AR2:287–288
defined, AR2:287
Dialog view (Connections Disk page),
Discontiguous selection, UI:282–283
Disk
Bookshelf, UI:109
browser, AWG:96
browsing, UI:105
Connected, UI:106–113
close, UI:106
collection, UI:115
directory icons, UI:73
form, AR2:301
full, UI:73, 77
content, UI:78
management, UI:105
directory line for, UI:107
network, UI:114–115
option sheet, UI:112
quick install, UI:113
reference, PDT:27–28
saving format information, PDT:45
viewer, AWG:32
Disk-based bookshelf, UI:13–14
Disk-based operating system, UI:10–11
Diskless robot, AWG:7
Display
commands, PDT:85–86
embedded objects and, UI:166–167
parameters, UI:274
profile information, PDT:119–120
size, UI:153
Displaying
captured scribbles, AR1:557–558
insertion pads, AR1:587
Directory
directories, AWG:28; AR1:89; AR2:43
application, AR2:391–395
bitmap, PDT:207–208
character, PDT:208
concepts, AR2:382
creating, AR2:69–74
document, AR2:141
deleted, AR2:52, 54
deleting, AR2:75
with browser, AR2:141
Desktop, AWG:33
directory entries and, AR2:54
document, PDT:55
directory tree node, AR2:93
Empty Application, PDT:52
forcing deletion of, AR2:75–76
Help Notebook, AR2:179–180
item, AR2:376
locating, AR2:56
merge, UI:72
method flags, AR2:71
name clash and,
name, AR2:52
names, AWG:29
names of, AR2:70–71
Notebook, AWG:33
renaming, AR2:141
root, AR2:52
handle, AR2:60–61
section, AWG:34
service, AR2:395–396
source, PDT:77
target, AR2:59
changing, AR2:86
see also specific directories
Directory entry, AR2:54
reading, AR2:87–89
all, AR2:88
sorting, AR2:88–89
Directory handles, AR2:43, 59–61
creating, AR2:58, 60, 71
directory nodes and, AR2:59
instance messages, AR2:64
locators and, AR2:59
observing, AR2:60
RAM, AR2:61
target directory and, AR2:59
using, AR2:60
volume root, AR2:60–61
well-known, AR2:60
working, AR2:61
see also Directories
Directory icon, UI:108
Directory index, AR2:56, 67
creating and using, AR2:80–81
Directory view (Connections Disk page),
Ui:108
Layout sheet, UI:110
Options menu, UI:111
Digitizing tablet, configuring, PDT:45
Direct manipulation, UI:150
digitizing tablet, configuring, PDT:45
Digitizers, PDT:56
Device
Digitizing tablet, configuring, PDT:45
Dialog sheets,
Device List (Show menu),
device-independent RGB method,
Dial type, modem, AR2:284
Dialog sheets, UI:40, 162, 205–206
command buttons, UI:178
for installable accessory instances,
UI:163
for installable stationary, UI:162
layout guidelines, UI:206–209
devices, UI:209
non-standard layout, UI:206
pop-up vs. in-line, UI:207–208
standard layout, UI:206
modal, UI:165, 205–206
modeless, UI:205
advantages of, UI:205
multi-page, UI:43
multiple, UI:205
pop-up lists, UI:183
Quick Help and, UI:215
single, UI:205
titles, UI:205
uses, UI:205
vertical checklists, UI:182
Dialogs, system and application, AR1:493
Dial string modifiers, AR2:287–288
defined, AR2:287
function, AR2:287–288
Dial type, modem, AR2:284
Dictionary
Bezier, PDT:210
icon, UI:78
installable software views and, UI:109
installed, UI:105
template, UI:242
Digitizers, PDT:56
title line, UI:47
window messages, AR1:163
wrapper, AR2:310
see also Documents

Documentation, PDT:9-23
feedback on, PDT:10
SDK library, PDT:9
suggested approach to, PDT:9-10
using fonts in, PDT:198-199
see also SDK, documentation

Document hierarchy messages,
AR1:161-163
application’s name, AR1:163
application’s title, AR1:163
document information, AR1:161-162
document links, AR1:161
embedded documents, AR1:161
hot mode, AR1:162
renaming document, AR1:162

Documenting applications, AWG:175

Document menu, UI:57-61; AR1:165;
AR2:102-107
About, UI:194
About, UI:194
CheckPoint, UI:193
Print Preview, UI:193-194
Print Setup, UI:193-194
Revert, UI:193
Embedded Printing sheet, UI:60-61
Export to Home command, PDT:170, 171
guidelines for using, UI:192
Headers & Footers sheet, UI:59
MiniText, AR1:363
Print command, AR1:139
Print Layout sheet, UI:58
Print Setup button, AR1:138
Print sheet, UI:57
standard application, AR2:331
table of contents, UI:85
uses, UI:192

Document oriented design, AWG:17

Documents, AWG:23; UI:14; AR1:82, 87-92
activating, AWG:23, 36-37, 105;
AR1:102-107
main in, AR1:102-103
msgAppActivate in, AR1:105-107
msgApplnit in, AR1:105-107
msgAppMngrActivate in, AR1:103
msgInit in, AR1:104
active, AR1:87
application hierarchy and, AWG:28
applications as, UI:157-158
basic layout for, UI:55
closing, AR1:109-110
components for, UI:56
illustrated, UI:55

compound, UI:18
cork margin, UI:20
creating, AWG:105; AR1:102, 148
with browser, AR2:141
new, UI:86-87; PDT:54
default menus for, UI:56
default names of, AWG:103
defined, AR1:67
deleting, AWG:38; AR1:115
directories, AR1:89; PDT:55
disk Bookshelf, UI:109
email, UI:121
embedded, UI:17; AR1:67, 117-128
Apply To control, UI:60
borders and controls for, UI:19-20
counting, AR1:183
creating, UI:20
directory, AR1:89
getting and setting number of,
AR1:181-182
icons and, UI:73
in-line, UI:169-170
pop-up, UI:169-171
printing, UI:60-61; AR1:136-137
special handling of, UI:168-169
traversal and, AR1:129
up-down gesture and, UI:66
embedding, AWG:34-35; AR1:77
Empty Application, AWG:95
copying to hard disk of, AWG:97
corkboard margin for, AWG:97
embedding, AWG:97
floating, AWG:95
messages, AWG:97
tabs for, AWG:96
fax, UI:121, 165
files/applications and, AWG:37-38
file system and, AR1:88-89
as floating accessories, AWG:104
global sequence number, AR1:178
help, UI:124
icons for, UI:73
application, UI:216-217
list of, UI:77
In box, UI:13
options for, UI:164
incoming, UI:121
life cycles, AWG:23; AR1:100-115
marking locations within, UI:171-172
Menu box (Stationary notebook),
UI:120
multi-page, UI:157-158
multiple, table of contents and, UI:86
name clash notes and, UI:71
names of, AWG:33
notebook page and, UI:157
as Notebook pages, AWG:104
as objects, AR1:92
with objects and frames, AR1:107
off-screen, AWG:24
open, AR1:67
process, AR1:109
Drivers
universal pen, PDT:56–62
using UniPen, PDT:60–62

DTR (data-terminal-ready) lines, AR2:270

Dual command path, UI:15–16
for application functionality, UI:224
borders and, UI:20
design checklist and, UI:295
edit pad design and, UI:189
for switching modes, UI:246
toggle gesture and, UI:247
for zooming, UI:265

Dumping, objects, AWG:161
Duplex mode, modem, AR2:286
DVHSPKT, AR2:274

Dynamic drawing, AR1:293
Dynamic Link Libraries. see DLLs
Dynamic ports, AR2:296
Dynamic resource IDs, AR2:343–344
defined, AR2:343
Dynamic UIDs, AR1:9, 10
DynResId() macro, AR2:345

EDA (embedded document architecture), AWG:12, 13

Edit function, UI:16

Editing
bitmaps, PDT:192–195
color shapes, PDT:182–190
fill-in fields, UI:34
font header, PDT:196–198
graphics, AR1:324
Help notebook index entries, UI:125
hints, PDT:190
ink, UI:255
guidelines for, UI:255–259
outline, window, PDT:183–184
overwrite fields and, UI:33, 34
picture segment, AR1:324
pixels, PDT:194
scrolling lists, UI:187

text, AR1:321

seeing also Edit pads

Editing model, AWG:11
AR1:165; PDT:171
application-specific, UI:195
Checkpoint command, PDT:170
choosing editing mode from, PDT:171
Copy command, AR1:120
creating object guidelines, UI:195
customized, UI:195
default, UI:194
default commands, UI:56, 62–65,
194–194
Undo, UI:194
drag & drop operation and, UI:68
editing operations and, UI:16
tables, PDT:171
Find command, UI:62, 63
Find sheet, UI:62
guidelines for using, UI:192
for initiating drag, UI:286
MiniNote, UI:136–137
Move command, AR1:120
Proof sheet, UI:65
Spell sheet, UI:64
table of contents, UI:86
uses, UI:192

Edit pads, UI:189–191
customizing, UI:189
designing, UI:189
input behavior, UI:190–191
invoking, UI:189
keyboard focus of, UI:189
modeless, UI:190–191
pop-up, UI:47, 189–190

see also Editing

Ellipse convention, button labels, UI:175
dragging, AR1:272
drawing, AR1:341

Email
documents, UI:121
service, UI:164

Embedded application, AWG:34–35;
AR1:77, 195
creating, AR1:196

Embedded chart, UI:167

Embedded Document Architecture
(EDA), UI:17–20
designing for, UI:152–153
marking document locations and,
UI:171–172

Embedded documents, AWG:28; UI:17;
AR1:67, 117–128
activating, AR1:161
Apply To control (Embedded Printing sheet), UI:60
borders and controls for, UI:19–20
compound documents and, UI:18
counting, AR1:183
creating, UI:20
design checklist and, UI:295
directory, AR1:89

Embedded window concepts, AR1:117–118
moving and copying between,
AR1:118–119

Empty Applications and, AWG:97
gestures and, AR1:128
getting and setting number of,
AR1:181–182
icons and, UI:73
illustrated, UI:18
in-line, UI:169–170
intercepted messages and, AR1:127
managing, AR1:161
move/copy protocol and,
AR1:119–123
moving in Tic-Tac-Toe and,
AR1:123–127
pop-up, UI:169–171
printing, UI:60–61; AR1:136–137
Location control, UI:60
situations for, AR1:136
selection and, AR1:128
special handling of, UI:168–169
traversal and, AR1:129
up-down gesture and, UI:66

see also Container application classes;
Documents

Embedded icons, UI:168
in-line, UI:168
pop-up, UI:170–171

see also Icons

Embedded objects, UI:152; PDT:18
displaying, UI:166–167
layout of, UI:168–171
in-line opening, UI:169–171
open document handling,
UI:168–169
managing, UI:166–172
basic support for, UI:166–167
intelligent layout, UI:168–171
marking document locations and,
UI:171–172

storing, UI:166
tracking, UI:166
in views, AR2:26–27
window, AR2:161

see also Objects

Embedded Printing card, AR2:136
Embedded Printing sheet, UI:60–61
Apply To control, UI:60
Location control, UI:60
Print control, UI:60

Embedded window, AR1:117
child, AR1:193
concepts, AR1:117–118
csEmbeddedWindow descendants,
AR1:118
marking support, AR1:118
error, PDT:130
exit, PDT:130
fault, PDT:130
generation, AR1:563
handling, AR1:563–564
inserting, AR1:550
intReq, PDT:130
keyboard, AR1:581–584
data, AR1:581–584
low-level pen, AR1:559–560
messages, AR1:546–547
oc, PDT:131
pen, AR1:575–580
processing, AR1:563–566
pen input sampling and,
AR1:565–566
x-y distribution and, AR1:565
program, PDT:129
queue, inserting messages in, AR1:571
routing, AR1:547
serial, AR2:266
break status, AR2:273
detecting, AR2:272–273
mask indicators, AR2:272
polling for, AR2:273
status codes, AR1:564
task, PDT:129–130
Exacting-once, datagram delivery, AR2:296
Examining
font header, PDT:196–198
text samples, PDT:198
Exception handling, PDT:148
Exclusive access services, AR2:445–446
defined, AR2:445
Executing
application, PDT:54–55
C code, PDT:79–80
Execution
control commands, PDT:85
controlling threads of, PDT:124
skipping, PDT:123–124
Exit events, PDT:130
Exiting, PenPoint, PDT:55
Expand command (table of contents
View menu), UI:86
Explicit locators, AR2:56
Explicit modes, UI:244–245
Export
note, UI:70
option card, AWG:172
EXPORT_DOC structure, AR2:154
Exported names, AWG:72
EXPORT_FORMAT structure, AR2:153, 154
Exporting, UI:69
bitmap, PDT:169–170
Connections notebook and, UI:105
documents, UI:108
option card, PDT:172
in PenPoint, UI:69
Exporting files, AR2:147
application responsibilities, AR2:150
clsExport messages, AR2:152–154
msgExport, AR2:154
msgExportGetFormats,
AR2:152–153
msgExportName, AR2:153
export dialog, AR2:149
export overview, AR2:148–149
file export mechanism, AR2:147
how export happens, AR2:152
see also Importing files
EXPORT_LIST structure, AR2:153
Export to Home (Document menu),
P DT:170
Expression handling macros, PDT:139
Extensibility, AWG:14–15
Extensions, AWG:70
Extracting
window, AR1:218, 234
from window tree, AR1:550
Facilities
clsPaper, AR1:590–592
examples, AR1:591–592
for networking and connectivity,
AR2:250
Failed() macros, AR1:24
Failures, during msgInit/msgRestore,
AWG:153
Fault events, PDT:130
Fax
documents, UI:121, 165
service, UI:164–165
Viewer application, AR1:317
FCNLTH, AR2:114
FEDIT: see Font editor (FEDIT)
Feedback
audible, UI:213–214
message line and, UI:214
selection, UI:187, 278–279
visual, UI:247
F gesture, UI:27, 63
MiniNote, UI:142
MiniText, UI:134
Field, AWG:123; AR1:475–486
activation and deactivation,
AR1:481–482
messages, AR1:481
Adobe Type 1 font, PDT:200
component creation, AR1:480–481
creating, AR1:477–479
custom handwriting translation,
AR1:479
style flags, AR1:477–479
data-specific, AR1:486
delayed input, AR1:483
font attribute window, PDT:198
font header window, PDT:197
input processing, AR1:482–483
input validation, AR1:484–485
layout, AR1:485
menu, UI:199
messages, AR1:476–477
properties access, AR1:480
sample, AR1:475
style flags, AR1:475–476
edtType settings, AR1:478
focusStyle setting, AR1:486
support, AR1:416
translator, AR1:479
user interface and, AR1:485
see also clsField; Fill-in fields;
Overwrite fields
FIELD_MOVEONLY structure, AR1:477
FIELD_STYLE, AR1:477
flags, AR1:479
values, AR1:478–479
Figure drawing operations, AR1:271–272
closed figures, AR1:271–272, 294
defined, AR1:269
open figures, AR1:271, 293
File attribute arguments, AR2:76
File browser, AWG:13
File commands, PDT:86
Filed state, AWG:35
File folder, directory icon, UI:108
File handles, AR2:43, 61–62
access intentions, AR2:62
byte position and, AR2:61
creating, AR2:71–73
file access control and, AR2:62
instance messages, AR2:64
locators and, AR2:59
translating file pointer into, AR2:66
File icon, UI:78, 108
File import and export, AR2:147–154
clsExport messages and, AR2:152–154
clsImport messages and, AR2:150–152
concepts, AR2:147–150
functions, AR2:124
interface, AR2:252–253
mechanisms, AR2:147
see also File System
File menu (FEDIT), PDT:179, 180
Font Attribute command, PDT:196
Font Header command, PDT:196
Save Subset command, PDT:196
File pointer, AR2:66
Files, AR2:43
access control, AR2:62
Adder, AWG:261
Adobe Type 1 font, PDT:199
Basic Service, AWG:272
Calculator, AWG:262
Clock, AWG:264
closing, AWG:143; AR2:46, 74–75
with stdio, AR2:66
Counter Application, AWG:196
creating, AR2:69–74
debug stream data, PDT:37
defined, AR2:55
deleting, AR2:75
with browser, AR2:141
DLC, AR2:401; PDT:53
DLL, AR2:399; PDT:53
creation options, AR2:402–403
DLC files and, AR2:401
DLLMain() routine and, AR2:400
issues, AR2:400
MAKE files and, AR2:403–404
operating system, AR2:403
references to, AR2:403
sharing, AR2:401–402
unloading, AR2:400
versions and, AR2:402
Empty Application, AWG:91, 177–178
font, editing, PDT:179–180
forcing deletion of, AR2:75–76
format compatibility of, AWG:17
header comments for, AWG:73
Hello World (custom window), AWG:188
Hello World (toolkit), AWG:181
initialization, PDT:28
Inputapp, AWG:270
locations of, AR2:67
MAKE, AR2:403–404
map, PDT:148
memory-mapped, AWG:143; AR2:55
MILL Service, AWG:274
mode flags, AR2:72
names of, PDT:52
checking, AR2:70–71
file system and, AWG:7
STAMP and, AWG:94
Notepaper App, AWG:265
opening, AWG:37–38, 143; AR2:46
for the first time, AWG:143–144
to restore, AWG:144–145
with stdio, AR2:66
organization of, AR2:381–398
Paint, AWG:267
PENPOINT.DIR, PDT:55
position and size of, AR2:84–85
protecting data, AR2:66–67
reading, AR2:83
from, AWG:142
registering types of, AR2:151
renaming, AR2:141
resource, AR2:337, 342
compacting and flushing, AR2:353
definition, AR2:355
organization, AR2:355–356
viewing contents of, AR2:359, 360
SDK, AWG:67
S-Shot, PDT:177–178
swap, PDT:39
error messages and, PDT:47
SYSCOPY.INI, PDT:43
table data, AR2:214
temporary, using handles with, AR2:65
Test Service, AWG:273
Tic-Tac-Toe, AWG:205–206
TIFF, PDT:175
Toolkit Demo, AWG:269
used in DB session, PDT:69
WATCOM Make, PDT:148
Wriptap, AWG:272
writing to, AWG:141; AR2:83
see also File Handles; Header files
files command, PDT:97
File system, AWG:72–76
coding suggestions, AWG:76
comments, AWG:75
defines, types, globals, AWG:74
file header comment, AWG:73
function prototypes, AWG:74
include directives, AWG:73–74
indentation, AWG:75
message headers, AWG:75
Field system, AWG:7–8; PDT:18
accessing, AR2:57–68
with stdio, AR2:65–66
attributes, AR2:54–55, 77
auxiliary notebooks and, AR2:422
browser, AR2:124
class and object use in, AWG:20
classes, AR2:58
subclassed, AR2:67
common, operations, AR2:47
concurrency considerations and, AR2:66–67
connectivity and, AR2:244
developer’s quick start, AR2:44–46
directories, AWG:28; AR2:54
documents and, AR1:88–89
document state and, AWG:39
embedded windows and, AR1:122
directories, AR2:55
functions, AR2:43–44
handles, AR2:43, 57–62
directory, AR2:59–61
file, AR2:61–62
functions, AR2:57
locators and, AR2:58–59
using with temporary files, AR2:65
hierechy, embedded applications
and, AWG:35
interface, AR2:252
locators, AR2:55–56
making, changes, AR2:141–142
messages, AR2:62–64
nodes, AR2:52–54
accessing, AR2:57
names, AR2:53–54
service instance, AR2:443
Notebook hierarchy, AWG:31
Notebook use of, AR2:68
organization, AWG:28–29; AR1:88–89
overview, AR2:43–44
paths, AR2:430–431
constants, AR2:430
Penpoint comparison with other systems, AR2:46–47
PENPOINT.DIR file and, AR2:68
performing, operations, AR2:43
principles and organization, AR2:49–56
programmatic interface provisions, AR2:57
services and, AR2:443–445
structure, AR2:43
using, AR2:69–91
changing target directory, AR2:86
closing files, AR2:74–75
comparing handles, AR2:86–87
copying and moving nodes, AR2:80–81
creating directories and files, AR2:69–74
deleting files and directories, AR2:75
ejecting floppies, AR2:91
file position and size, AR2:84–85
flushing buffers, AR2:85
forcing deletion of files/directories, AR2:75–76
getting and setting attributes, AR2:76–80
getting path handle, AR2:85–86
getting volume information, AR2:90–91
handle mode flags, AR2:87
making native node, AR2:89–90
node existence determination, AR2:83
observing changes, AR2:89
reading and writing files, AR2:83
reading directory entries, AR2:87–89
renaming nodes, AR2:83
setting/changing volume name, AR2:91
traversing nodes, AR2:81–82
volume specific messages, AR2:91
volumes, AR2:49–52
see also File import and export
Filing, AWG:28
controls, AR1:399
counter object, AWG:145–146
DC, AWG:132–133
disadvantage of, AWG:143
frames, AR1:503
list boxes, AR1:469
methods, AWG:143
object, AWG:129–132, 140–142
state, AWG:135
UI Toolkit components, AR1:365
windows, AR1:253
Fill command (Edit menu), PDT:171
Filled region (progress bar), AR1:531
defaults, AR1:537
manipulating, AR1:537–539
FILLED.TXT, AR2:260
Fill patterns, AR1:33
checklists with, UI:186
editing gestures for, UI:34
fonts, UI:188
guidelines for using, UI:188
menus with, UI:41
edit pad, UI:199
scaling, UI:268
size of, UI:188
toggle switches and, UI:180–181
visual segmentation cues and, UI:241
see also Overwrite fields
Fill patterns, AR1:291
alignment, AR1:292
determining, AR1:292
graphic state element, AR1:278
styles, AR1:292
windows and, AR1:294
Filter, AR1:548–549
adding, AR1:571
objects, AR1:547
removing, AR1:571
FIM_GET_INSTALLED_ID_LIST structure, AR2:418–419
FIM_GET_NAME_FROM_ID structure, AR2:418
FIM_PRUNE_CONTROL structure, AR1:441
FIM_SHORT_IDs, AR1:441
Find & Replace sheet, UI:63
Find button, UI:63
Replace button, UI:64
Search button, UI:63
Find command (Edit menu), UI:62
mark protocol and, UI:172
Find sheet, UI:62
modifying and replacing, UI:194
confirmed command, PDT:98
FIXED numbers, AR2:115
Fixed-point calculations, AR2:115
functions, AR2:116–117
numbers, AR1:288; AR2:115
Flags, AWG:79
built-in rule, AR1:600–601
checking, AWG:112
clsBorder style, AR1:373–375
in code profile syntax, PDT:114
constraint, AR1:395
debugging, AWG:32, 85–86
described by APP_MGR_FLAGS, AR1:147
directory mode, AR2:71
dp, PDT:119
existence, AR2:70
field style, AR1:475–476
file mode, AR1:180; AR2:72
filter, AR1:298
FS_SEEK, AR2:84
getting and setting, AR1:180
handle mode, AR2:87
handwriting translation, AR1:600–602
input, AR1:558, 565–566
clsSPaper, AR1:594–595
window, AR1:569–570
knowledge source, AR1:601–602
node attributes, AR2:79–80
note, AR1:489
object capability, AR1:25
in object profile syntax, PDT:117
scrollwin, AR1:459
setting values to, AWG:86
table layout, AR1:384–385
toolkit table, AR1:430–432
modifying items with, AR1:435
values, AR1:431
transaction item, AR2:203–204
TV_STYLE, AR2:25–26
window, AR1:228–229
input, AR1:228
setting, AR1:234–235
style, AR1:228, 229, 232
XList, AR1:611
element, AR1:612
Flag sets, debugging, PDT:134–135
f command, PDT:97
file formats, PDT:180
Flick gesture, UI:16, 24
Double, Triple, Quadruple, UI:25
guidelines for, UI:239
family, UI:24
summary, UI:235
in gesture mode, UI:258
guidelines for, UI:237
hot point for, UI:231–232
MiniNote gesture mode, UI:135, 141
MiniNote ink mode, UI:135
MiniText, UI:133–134
pop-up checklists and, UI:30
scrolling with, UI:264–265
to switch modes, UI:249
Tap & Flick and, UI:37
see also Gestures
Flick Left-Right gesture, UI:26
guidelines for, UI:238
ink editing, UI:258
MiniNote, UI:140
to toggle modes, UI:256
Flick Up-Down gesture, UI:26, 66
guidelines for, UI:238
Float & Zoom preferences (Settings notebook), UI:94
Flooding, AWG:95
accessories, AWG:34, 104
navigation control area, UI:267
page, UI:15
Floating allowed preference, AR2:365
Floppy disk
inserting, AR2:91
icon, UI:77
Flow control, AR2:265–266
characters, AR2:270
protocols, AR2:265
specifying, AR2:269
using, AR2:272
Flow pagination, AR1:137
Flushing
buffered output, AR2:353
buffers, AR2:85
input and output, AR2:271
resource files, AR2:353
streams, AR2:136
Folder icon (open/closed), UI:78
FONT directory, AR2:384, 386
Font editor (FEDIT), PDT:179–210
Adobe Type I fonts, PDT:199–204
concepts, PDT:179–182
editing bitmaps, PDT:192–195
editing character shapes, PDT:182–190
editing hints, PDT:190–192
font file formats, PDT:204–210
function, PDT:179
getting started with, PDT:179–180
miscellaneous functions, PDT:195–199
Font Field preference, UI:188
Font icon, UI:78
Font installation manager, AR2:416–419
FONTDBNEWONLY structure, AR1:473
Font list boxes, AR1:473
creating, AR1:473
notification, AR1:473
Font menu (System Log), PDT:143
Fonts & Layout preferences (Settings notebook), UI:93
Fonts, UI:101; AR1:275–276
adding character to, PDT:193
aspect ratio of, AR1:306
attributes, AR1:275, 303, 305
attribute window, PDT:197
fields, PDT:198
bitmaps, AR1:275, 314; PDT:192
cache loading, AR1:314
changing, AR1:302
common, AR1:306–307
to switch modes, UI:249
control of (MiniNote), UI:138
default, AR1:305–306
defined, AR1:303
encoding, AR1:307
fields, AR1:312
enumeration of, AR1:304
file
creating, PDT:180
ingesting, PDT:179
file formats, PDT:204–210
Nimbus-Q, PDT:204–206
PenPoint Packed format,
PDT:206–210
geometry, AR1:308–309
gesture, AWG:174
Gesture, file, AR2:188–190
graphic state element, AR1:278
groups of, AR1:305–307
handle, AR2:416
finding, AR2:418
header
editing and examining,
PDT:196–198
Nimbus-Q, PDT:205
PenPoint Packed, PDT:206–207
window, PDT:196
window fields, PDT:197
ID, AR1:275, 276, 303
opening fonts and, AR1:303–304
identification, AR2:416–417
IDs, getting and setting, AR2:418
installable software views and, UI:109
installed, UI:101
displaying, AR1:441
list of, AR2:418–419
installing, AR2:378
metrics, AR1:275, 307–310
names, AR1:303, 308; AR2:418
opening, AR1:275–276, 303–305
outline, AR1:275; PDT:16
 scales, AWG:127
scaling, AR1:276, 310–311
search path, AR1:313–314
state, AR1:315
storing, AR1:304
strings, AR1:304–305
switching between, AR1:315
system, AR1:306; AR2:363
scaling with, UI:269–270
size of, UI:274
transforming, AR1:307
units, PDT:181
user, AR2:363
choice of, AR1:304
user-visible, AR1:306
using, in documentation,
PDT:198–199
weight of, AR1:306
Font size
preference, UI:188
system, UI:269–270
user defined, UI:271
FOO.C, AWG:252, 257–259
FOO.H, AWG:252, 256–257
Foreground colors, AR1:274
graphic state element, AR1:278
painting, AR1:376
palette colors, AR1:295
RGB color values, AR1:295
Format codes, AR1:495–496
FORMATFILE, AWG:144
Frame application, AR1:212
Frame decorations, AR1:141, 507–509
close box, AR1:507
command bar, AR1:508
page number, AR1:509
removing, AR1:140–141
shadow, AR1:509
tab bars, AR1:508
title bars, AR1:507
user interface, AR1:507
frameNewFields, AR1:16
FRAME_NEW structure, AR1:19
FRAME_ZOOM structure, AR1:500
Frame numbers, PDT:75
Frame object, AWG:66
Frames, AWG:227, 113; AR1:497–505;
PDT:16
Application Framework and,
AR1:504–505
client windows and, AWG:122
components of, AR1:497
creating, AWG:121; AR1:500
custom layout example, AR1:389
filing, AR1:503
items included in, AWG:120
layout, AR1:500–501
messages, AR1:499
modifying, AR1:500
multiple, AR1:500
notification, AR1:501–503
close, float, bring-to-front, delete,
AR1:503
selection, AR1:502
zoom, AR1:502–503
resizing and dragging, AR1:501
scrollbars and, AR1:498
styles of, AR1:498
subclasses of, AR1:505
Toolkit window inside, AWG:121
see also clsFrame
FRAME STYLE structure, AR1:500
FRAME_ZOOM structure, AR1:503
Freeing capability, AR1:26
Freeze count, PDT:124
thawing, PDT:124
FS_CHANGE_INFO structure, AR2:89
fs command, PDT:98
for enabling logging, PDT:148
for setting debug flags, PDT:147
fsDirNewDefaultMode, AR2:71
FS_DIR_NEW_MODE, AR2:71
FS_EXIST constants, AR2:70
fsExistGenUnique flag, AR2:70
fsFileNewDefaultMode, AR2:73
FS_FILE_NEW MODE constants, AR2:72
FS_FLAT LOCATOR structure, AR2:140, 141
FS_FORCE_DELETE structure, AR2:75
FS_GET_PATH structure, AR2:85–86
FS_GET_SET_ATTRIB structure, AR2:76–77
getting values and, AR2:78
setting values and, AR2:79
FS_GET_VOL_METRICS structure, AR2:90
FS.H, AR2:69
attribute label macros, AR2:77–78
FS LOCATOR structure, AR2:86
FSMakeAttr macro, PDT:163
FS_MAKE_NATIVE structure, AR2:90
FS_MOVE_COPY structure, AR2:80
FSNameValid() function, AR2:53, 70
FS_NEW ONLY structure, AR1:179
FS_NEW structure, AR1:179; AR2:69–70, 71
FS_NODE_EXISTS structure, AR2:83
FS_NODE_FLAGS structure, AR2:79
fsNoExistCreateUnique flag, AR2:70
fsSharedMemoryMap, AR2:73
tempFile, AR2:65
FS TRAVERSE structure, AR2:81
FSUI task, PDT:153
FS_VOL_METRICS structure, AR2:61
fullEnvironment flag, AR1:98
Functions, AWG:71
message passing, PDT:138
prototypes of, AWG:74
qualifiers, AWG:78
FxMakeFixed() routine, AR2:115
fs command, PDT:124
Gauges, UI:38
Gazelle, sample definition, PDT:61
g command, PDT:71
code addresses and, PDT:88
datasheet, PDT:99
GDIR command, AR2:390
GDIR utility, PDT:162
Generate Mask command (Edit menu), PDT:171
Geometric shapes, rendering with thick borders, AR1:340–342
Gesture recognition engine, UI:233

Gesture, font, AR2:188–190

Gesture icon, UI:78

Gesture margin, UI:135–136, 257

Gesture mode, UI:245–246

core, UI:247

guidelines for editing in, UI:257

MiniNote, UI:135

toggle switch, UI:30

default, UI:248

see also Gestures

Gesture recognition engine, UI:233

Gestures, AWG:4; UI:15–17, 23–27

to Quick Help strings, AR2:191

adding, to Quick Help strings, AR2:191

bounding box, UI:231

categories of, UI:23

capital letter accelerators, UI:27

core, UI:16–17, 24–25

non-core, UI:23, 25–27

collisions of, UI:233–234

for drag & drop operation, UI:68

dual command path and, UI:15–16

embedded window, AR2:127

dragging and dropping, AR2:165

for zooming, UI:266

see also specific gestures and types of gestures

Gesture timeout preference, AR2:364

Gesture windows, AR1:368–370

gesture propagation, AR1:368–369

messages, AR1:617–618

Quick Help IDs, AR1:370

responding to gestures, AR1:369

using, AR1:617–618

see also clsGWin

Getting and setting

application directory attributes, AR1:180–182

all, AR1:180

individual, AR1:180–182

embedded window style, AR1:190–191

pixel values, AR1:259

XList elements, AR1:615

Globals, AWG:74

file section, AWG:74

Global sequence numbers, AR1:178

generating, AR1:179

Global well-known UID, AWG:53

Glyphs

finding, AR1:313–314

metrics and encoding, AR1:314

search path, AR1:313–314

missing, AR1:277

GO Address book, AR2:323–324

gestures, AR2:323

illustrated, AR2:324

loading, AR2:323

using, AR2:323

see also Address book

GO.BAT, PDT:45

defined, PDT:165

modifying for, PDT:45

GO Fax icons, UI:77, 216

GO.H, AWG:77

bit definition, AWG:79

compiler independence and, AWG:77

uppercase keywords and, AWG:78

GO's proprietary packed format. see PenPoint, Packed format

Goto buttons. see Reference buttons

Grabber, AR1:549

input, AR1:572

object, AR1:547

Grab boxes, AR1:528–529

Grafics, AR1:319–320

coordinates, AR1:320

current, AR1:324

defined, AR1:319

drawing by adding, AR1:325

drawing messages and, AR1:319

editing, AR1:324

index, AR1:324

invisible, AR1:326

opcode, AR1:319

scaling, AR1:327

Graphical gestures. see Gestures

Graphic design

fundamentals, UI:149

developing, UI:179–182

of icons, UI:219–222

Graphics, AWG:9

classes, AR1:210–211

hierarchy, AR1:211

coordinate systems, AR1:267–269

drawing context features, AR1:270

coordinate systems, AR1:267–269

drawing context features, AR1:270

models and implementation, AR1:265–267

overview, AWG:128–129

subsystem, AR2:5; PDT:18

using picture segments in,

AR1:323–326

see also ImagePoint; System drawing context

Graphic state, AR1:270, 277–278

determining filled areas, AR1:292

elements, AR1:277–278

filling and stoking, AR1:290

fill pattern alignment, AR1:292

line and fill patterns, AR1:291–292

line styles, AR1:290–291

messages, AR1:281

raster operations, AR1:292–293

setting attributes of, AR1:271

storage, AR1:277

Grid

for aligning controls, UI:206

building translators with, UI:241

use in graphic design, UI:149

Group command (MiniNote Arrange menu), UI:137

Group, font, AR1:305–307

common, AR1:306–307

default, AR1:305–306

encoding, AR1:307

names, AR1:305

transforming, AR1:307

growChildHeight flag, AR1:385, 387

growChildWidth flag, AR1:385, 387

growChildHeight flag, AR1:385, 387

GUI (Graphical User Interface), UI:149

button label conventions, UI:175

drag & drop model, UI:285

explicit modality and, UI:244–245

spacial modality and, UI:244

GWIn: AWG:167

GWIN_GESTURE structure, AR1:369, 617

see also

Graphical state, AR1:270, 277–278

determining filled areas, AR1:292

elements, AR1:277–278

filling and stoking, AR1:290

fill pattern alignment, AR1:292

line and fill patterns, AR1:291–292

line styles, AR1:290–291

messages, AR1:281

raster operations, AR1:292–293

setting attributes of, AR1:271

storage, AR1:277

Grid

for aligning controls, UI:206

building translators with, UI:241

use in graphic design, UI:149

Group command (MiniNote Arrange menu), UI:137

Group, font, AR1:305–307

common, AR1:306–307

default, AR1:305–306

encoding, AR1:307

names, AR1:305

transforming, AR1:307

growChildHeight flag, AR1:385, 387

growChildWidth flag, AR1:385, 387

GUI (Graphical User Interface), UI:149

button label conventions, UI:175

drag & drop model, UI:285

explicit modality and, UI:244–245

spacial modality and, UI:244

GWIn: AWG:167

GWIN_GESTURE structure, AR1:369, 617

see also

Half-outlined buttons, UI:177

Handle mode flags, AR2:87

Handles

comparing, AR2:86–87
creating, AR2:69–70
file system, AR2:57–62
directory handles, AR2:69–61
file handles, AR2:61–62
locators and, AR2:58–59
using with temporary files, AR2:65
finding, AR2:263–264
freeing, AR2:74–75
getting path of, AR2:85
objects, AR2:57
creating, AR2:58
on parallel port, AR2:251
on serial port, AR2:251
parallel port, AR2:276–277
resize, UI:272
serial, AR2:268
data modem, AR2:279–280
Hand preference, UI:35; AR2:363
Handwriting
engine, UI:100
gestures and, AWG:156–158
icon, UI:78
installable software views and, UI:109
installed, UI:99–100
processing, UI:241–243
constraining translation and, UI:241–243
using translators and, UI:241
profile, UI:100
translators, UI:241
building, UI:242
letters, numbers, punctuation, symbols, UI:242
Handwriting capture
classes, AR1:551
object, AR1:555
subsystem, AR1:555
Handwriting Preference sheet, UI:188
Handwriting timeout preference, AR2:364
Handwriting translation, AR1:479
engine, AWG:11
flags, AR1:600–602
built-in rules, AR1:600–601
knowledge source rules, AR1:601–602
post-processing rules, AR1:602
subsystem, AWG:4, 10; AR1:546, 551–553; PDT:17
translation classes, AR1:552–553
window subclasses, AR1:551–552
Hard disk icon, UI:77
Hardware, PC, PDT:25–28
labelling volumes, PDT:27–28
machine configurations, PDT:26
memory, caches, RAM disks, PDT:27
mouse, PDT:26
networks, PDT:27
specifications, PDT:25
Hardware requirements, UI:5
Hardware RTS/CTS flow control, AR2:265, 266
see also Flow control
h command, PDT:99
Header files, AWG:53; AR1:31; AR2:95
for class info, AWG:118
for clsApp, AR1:160–161
common, AWG:74, 102
creating, AR1:36
C source and, AWG:67
debugging flag sets and, PDT:135
enum value and, AWG:78
function prototypes, AWG:74
goto buttons, AR1:117
indentation, AWG:75
INTL.H, AWG:62
labels, AR1:14
libraries and, AWG:102
message, AR1:75
MT output file, AR1:46
multiple inclusion, AR1:73–74
structure, AWG:72
UI Toolkit classes, AR1:364
see also Files
Headers and Footers sheet, UI:59
Heaps, AR2:101–102
defined, AR2:101
management, AR2:101
size and characteristics of, AR2:102
HELLO.C, AWG:188–190
HELLO.DLC, AWG:195
HELLOTK.C, AWG:113, 181
code run-through for, AWG:114–122
highlights of, AWG:114–115
sample code, AWG:182–184
testing well-known UID, AWG:114
HELLOTK2.C, AWG:113, 181
custom layout window, AWG:123
enhancements, AWG:123
HELLOTK1.C comparison, AWG:122
highlights, AWG:122–123
layout, AWG:122–123
one client window per frame, AWG:122
sample code, AWG:184–186
HELLOTK, AWG:113
compiling and linking, AWG:114
tings to do with, AWG:114
HELLOWIN.C, AWG:190–194
HELLOWIN.H, AWG:190
HelloWinInit, AWG:131
code, AWG:131
Hello World (custom window), AWG:125–128
classes, AWG:187
clsHelloWin highlights, AWG:127–128
clsHelloWorld highlights, AWG:127
compiling, AWG:187
linking and, AWG:125–127
debugging, AWG:133–134
DLC files and, AWG:127
DLL files and, AWG:126
drawing in window, AWG:133
files, AWG:188
font scales and, AWG:127
modifying, AWG:133
objectives, AWG:187
page turn, AWG:132
running, AWG:187
sample code, AWG:187–195
tutorial, AWG:89
window creation, AWG:130
Hello World (toolkit), AWG:113–123
classes, AWG:117, 181
client window and, AWG:120
code run-through for HELLOTK1.C, AWG:114–122
compiling, AWG:181
creating application instances for, AWG:114
files, AWG:181
HELLOTK, AWG:113–114
installing, AWG:114
label creation, AWG:121
message sending, AWG:115–116
method table, AWG:114
msgAppInit, AWG:114–115
running, AWG:181
sample code, AWG:180–186
second HELLOTK highlights, AWG:122–123
testing, AWG:181
tutorial, AWG:89
UI Toolkit and, AWG:113
see also UI Toolkit
Help, UI:215; AR2:179–191
advanced topics, AR2:187–191
concepts, AR2:179–182
directory, AR2:393–394
documents, creating, AWG:165–166
gesture, AWG:166
Gestute font and, AR2:188–190
Help Notebook, AR2:179–180
icon, UI:13, 76, 123, 124; AR2:179
loading and unloading, AR1:154
message, UI:123
Notebook, AWG:13, 33; UI:13, 215;
AR1:154
Tic-Tac-Toe, AWG:165–166
Quick Help, AWG:166–168; UI:215;
AR2:181–182
resources, AR2:183–187
systems, AWG:13
templates, AR2:387
text, AR2:180
adding gestures to, AR2:190
creating, AR2:180
source files, AWG:250
HELP directory, AR2:387, 393–394
using, UI:254–259
ways of, UI:254–255

In-line fields, AR1:475

In-line style (embedded document), UI:168–170
dialog/options sheets and, UI:207–208
mixing with pop-up style, UI:207–208

Inner rectangle region, AR1:379

Input

delayed, AR1:483
messages, AR1:483
device drivers, AR1:567

INPUT_EVENT structure, AR1:567–568
distribution model, AWG:11

INPUT_EVENT_DATA structure, AR1:575–576
for keyboard events, AR1:581

InputEventGen() interface, AR1:549, 563

INPUT_EVENT structure, AR1:567

input flag, AR1:566
inputThrough flag, AR1:566

Input modes, UI:244–253
guidelines for using, UI:246–248
core gesture mode, UI:247
mode switch, UI:247
mode toggle switch, UI:247–248
modal applications, UI:253
role of, UI:244–246
switch location, UI:251–252
switch presentation, UI:248–250
user models for, UI:245–246

InputMoveDelta flag, AR1:565
inputNoBuy flag, AR1:566

Input pad style preference, AR2:366

Input subsystem, AR1:545, 546-550; AR2:27–29
API, AR1:567–573
event data structure, AR1:567–568
constants, AR1:568–570
event routing, AR1:547
filters, AR1:548–549
grabber, AR1:549
input registry, AR1:547–548
inserting events, AR1:550
listener objects, AR1:549
messages, AR1:570
procedures, AR1:571–573
adding filter, AR1:571
getting grab information, AR1:572
getting target, AR1:573
inserting message into event queue, AR1:571
removing filter, AR1:571
setting input grabber, AR1:572
setting input target, AR1:572
routing, AR1:548
target object, AR1:550
UIDs, AR1:568
window tree, AR1:550
inputTransparent flag, AR1:566
InRange macro, AWG:78

Insert function, UI:16

Inserting
character, AR2:12
custom window as main application
window, AR1:561–562
events in input stream, AR1:550
IP window, AR1:556–557
list box entries, AR1:467
messages in event queue, AR1:571
text view in scroll window, AR2:30
window in window tree, AR1:550
windows, AR1:217, 233–234
XList element, AR1:614

Insertion pads, AR1:585
creating, AR1:586–587
deleting, AR1:587
displaying, AR1:587
text, AR2:9
creating, AR2:33
destroying, AR2:33
embedding objects and, AR2:26
messages, AR2:33
using, AR2:33
UI styles, AR1:587

Insertion point, AR1:416

Insert Line command (MiniNote Edit menu), UI:136

Insert space function, UI:17

Inside AppleTalk, AR2:301

Installable applications, AR2:386–387
Installable entities, AR2:386

Installable items, AR2:411–412
altering, attributes, AR2:412
changing, name, AR2:412
deleting, AR2:412
duplicating, AR2:412
finding, AR2:413
getting and setting current, AR2:412
getting attributes of, AR2:413
getting information about, AR2:413
getting list of, AR2:413
installing, AR2:411
manager, AR2:410–411
size of, AR2:413

Installable services, AR2:387

Installable software
icons, UI:78
sheet, AWG:22

Installation, AWG:105
APIs, PDT:20–21
concepts, AR2:375–380
overview, AR2:573
classes, AR2:379–380
dialog, UI:113
configuration, UI:164
options, UI:165
document options and, UI:164
features, AWG:163
flexibility vs. simplicity, UI:165
initiation, AR2:376
manager class, PDT:21
MS-DOS, AWG:21–22
PenPoint, AWG:22
process, AR2:376–377
service, AR2:378–379, 450–456

Inputapp, AWG:269–270

INPUTAPP.C, AR1:559–560

Input buffer, AR2:265
flushing, AR2:271
status, AR2:271

see also Buffers
advanced clsInstallMgr topics and, AR2:414
application, AR2:415–416
code, AR2:414–416
font, AR2:416–419
installer concepts and, AR2:405–407
observing, AR2:407–409
service, AR2:416
using clsInstallMgr messages and, AR2:409–413
Install button (Settings notebook), UI:99
install command, UI:99
Installed Software section (Settings notebook), UI:13, 90, 97–101
Applications page, UI:98
Installed Dictionaries page, UI:100
Installed Fonts page, UI:101
Installed Handwriting page, UI:99–100
Installed User Profiles page, UI:101
install sheet, UI:99
menus, UI:98
options, UI:98
overview, UI:97–98
uses, UI:97–98
Installer, AWG:22; AR2:405
application, AWG:67
concepts, AR2:405–407
defined, AR2:406
deinstallation with, AWG:38
help documents and, AWG:165
PenPoint, PDT:20
in application installation,
PDT:50–51
correlation, PDT:63
functions, PDT:51
for installing while PenPoint is running, PDT:51
Quick, PDT:50
Quick Help and, AWG:168
responsibilities, AWG:22–23
Installing
application, PDT:50–53
boot-time install, PDT:52–53
to debug, PDT:70
.DLL and .DLF files, PDT:53
illustrated, PDT:52
while running PenPoint, PDT:51–52
application monitor and,
AR1:151–152
applications, AWG:21, 67; UI:13,
162–165; AR1:96–99;
AR2:377–378, 415
accessories, UI:163
document stationary, UI:162
flexibility vs. simplicity, UI:165
services, UI:163–165
classes, AR1:33, 47–48
new, AR1:146–147
CounterApp, AWG:137
DB, PDT:70
devices, AR2:307
from disk, UI:105
Empty Application, AWG:94
fonts and handwriting prototypes,
AR2:378
service class, AR2:441–442
services, AR2:307, 415
S-Shot, PDT:175
see also Adding
Install sheet, UI:99
Instance data, AWG:55, 130; AR1:33,
34–35
accessing, AWG:132; AR1:35
allocating, AR1:34
application, AWG:65
classes and, AWG:43
.clsApp and, AWG:120
.clsCntr, AWG:138–139
contents, AR1:99
CounterApp, AWG:142–146
defining additional, AR1:85
document activation and, AR1:104
Hello World (custom window),
AWG:130
instance info vs., AWG:153
maintaining dynamic, AR1:35
memory protection and, AR1:34
message handler and, AWG:60
modifying read-only, AWG:140
saving and restoring, AR1:35
saving in, AWG:65
size, AWG:54–55; AR1:47–48
updating, AWG:140, 142
using, AWG:132–133
Instances, AWG:103; AR1:5
accessory, UI:163
application, AR1:82
application classes and, AWG:24
classes and, AWG:43, 48
Class Manager messages and, AWG:106
classApp and, AWG:120
initializing new, AWG:49
objects and, AWG:43
processing of, AWG:47–48
of simple class, AR1:84
Tic-Tac-Toe, AWG:150
INST directory, AR2:396
service directory and, AR2:444, 445
Integral coordinates, AR1:268
Interaction checklists, AWG:68–69
Interfaces
connectivity and, AR2:244
data modem, AR2:279–293
devices and, AR2:249
file import/export, AR2:252–253
file system, AR2:252
high-speed packet I/O, AR2:252
modem, AR2:253
networking, AR2:253
parallel I/O, AR2:251–252, 275–278
serial I/O, AR2:251, 265–274
services and, AR2:249–250
SoftTalk, AR2:250
stream, AR2:246
Internal disk size, UI:103
Internationalization
data types and, AWG:77
defined, AWG:61
designing for, AWG:61–64
preparing for, AWG:63–64
Interrupts, PDT:148–151
Intertask communication, AR2:100–101
messages, AR2:100–101
semaphores, AR2:101
Intertask messages, AR2:100–101
Class Manager messages and, AR2:100
modes, AR2:100
processing order, AR2:100
INTL.H file, AWG:62; AR2:111
“Int w/o RB: 7” messages, PDT:47, 50
Invert command (Edit menu), PDT:171
IP_NEW typed structure, AR2:158
IP_XLATE_DATA structure, AR1:588
IsScaleFitWindowProper, AWG:119
Italic style, UI:16
Item directory, AR2:376
Items
controlling, AR2:406–407
installable, AR2:411–412
altering, attributes, AR2:412
changing, name, AR2:412
deleting, AR2:412
duplicating, AR2:412
finding, AR2:413
going and setting current,
AR2:412
going attributes of, AR2:413
going information about, AR2:413
going list of, AR2:413
installing, AR2:411
managers, AR2:410–411
size of, AR2:413
installed, database, AR2:406
installing, AR2:376–377
list
adding, AR2:129
counting, AR2:130
enumerating, AR2:130–131
getting, AR2:129
removing, AR2:130
removing all, AR2:130
replacing, AR2:130
NotePaper data, AR2:234–235
transaction data, AR2:201
contents, AR2:201
flags, AR2:203–204
Iterative development, UI:151

Jmp() macros, AR1:24

k command, PDT:100
Kernel, AWG:105; AR2:97
functions, AR2:105–107
date and timer routines, AR2:106
debugger entry routines, AR2:106
display/screen device routines, AR2:107
heap routines, AR2:107
intertask communications routines, AR2:106
keyboard routines, AR2:107
memory information routines, AR2:106
miscellaneous routines, AR2:107
task manager routines, AR2:105–106
tone routines, AR2:107
general protection fault handler, PDT:148
interface, PDT:18
layer, AR2:98
overview, AR2:97–107
semaphores and, AR2:101
services of, AR2:97
summary, AR2:105–107
task scheduler and, AR2:99
Kernel layer, AWG:6–7
defined, AWG:6
services, AWG:6
support features, AWG:6–7
Kerning, AR1:310
Keyboard
accessory
option sheet, UI:202
scales on resize, UI:273
event data, AR1:581–584
focus of edit pad, UI:189
handling, AWG:158
color, UI:13, 76
input and selection, AWG:153–155
PenPoint and, AWG:11
text selection and, UI:283
typing simulation, UI:13
KEYBOARD.H, AR1:581
keyCode, AWG:158
KEY_DATA structure, AR1:581–582
KEY.H, AR1:581
Keys, AR1:24
changing capabilities and, AR1:28
OBJECT_NEW_ structure, AR1:47
using, AR1:24–25
Key task, PDT:152

Keywords, AWG:78
asynchronous serial I/O, PDT:33
ENVIRO.INI, PDT:36
AutoZoom, PDT:36
BkShelfPath, PDT:37
BootProgressMax, PDT:37
Config, PDT:37
DebugLog, PDT:37
DebugLogFlushCount, PDT:37
DebugSet, PDT:38
PenPointPath, PDT:38
PenProxTimeout, PDT:38
ScreenHeight, PDT:39
ScreenWidth, PDT:39
StartApp, PDT:39
SwapMem, PDT:39
SwapBoat, PDT:39
SwapFileSize, PDT:39
TZ, PDT:39
Version, PDT:39
VolSel, PDT:40
WinMode, PDT:40
ZoomMargin, PDT:40
ZoomResize, PDT:40
MIL.INI, PDT:33–34
diagnosing information, PDT:33
disks, PDT:33
exit to DOS, PDT:34
high-speed packet parallel port
I/O, PDT:34
serial painting devices, PDT:33
TOPS FlashCard type, PDT:34
video controller, PDT:34
Wacom 310 digitizer, PDT:33

Knowledge source, AR1:601
rules, AR1:601–602
defined, AR1:600
spelling dictionary, AR1:601
translation template, AR1:601

LABEL.H, AWG:119
Labelling, volumes, PDT:27–28
LABEL_NEW_ONLY structure, AR1:411
label strings and, AR1:413
LABEL_RECTs structure, AR1:416
Labels, AWG:113, 114; AR1:409–416;
PDT:16
bold, UI:177
button, UI:175
wording for, UI:210–211
child windows and, AR1:415–416
clsCntrApp, AWG:138
creating, AWG:119; AR1:411–413
annotation, AR1:413
when to, AWG:121
for dialog and option sheets, UI:206
displays of, AR1:409
edit pads and, UI:47
field support and, AR1:416
for gauges, UI:38
HELLOTk.C and, AWG:114
icon, UI:74
layout, AR1:414
menu, UI:42, 177
for current mode, UI:249
messages, AR1:410
notification, AR1:414
painting, AR1:414–415
pop-up lists, UI:183
progress bar, AR1:531
custom, AR1:531, 539–540
reference button, UI:172
sample, AR1:409
strings and special characters, AR1:413
styles of, AR1:411–413
fields, AR1:411–412
suppressing, UI:183
for Tic-Tac-Toe, AWG:151
for Undo Edit command, UI:194
version, PDT:39
volume, PDT:40
see also clsLabel
LABEL_STYLE structure, AR1:411, 411–412

Language orientation, UI:276–277;
PDT:40

Language-dependent routines, AWG:64

Large icons, UI:74

LaserJet, AR2:247

Layer, AWG:6
application, AWG:6, 13
Application Framework, AWG:6, 12–13
component, AWG:6, 12
kernel, AWG:6–7
MIL (machine interface layer),
AWG:6, 273–274
system, AWG:7–12

Layout, AWG:122–123; AR1:361
automatic, UI:153–154
baseline, AR1:387
border window, AR1:378
calculator example, AR1:388
capturing vs., AR1:398
child windows, AR1:415–416
controls, UI:196
option and dialog sheets,
UI:206–209
custom, AR1:389–390
creating, window, AR1:390–395
dimensions, AR1:391–392
initialization, AR1:396
sample, AR1:389
embedded object, UI:168–171
open document forms and,
UI:169–171
open document handling and, UI:168–169
field, AR1:485
frames, AR1:500–501
icons, AR1:525
labels, AR1:414
landscape orientation, UI:276–277
lazy, AR1:397
loops, AR1:397–398
menu, UI:196–197
multi-column, UI:196
multiple, UI:163
non-vertical, UI:182–183
notes, AR1:492
optimizing, for small screens, UI:274–277
option card, AR1:515
orientation-specific, UI:276–277
portrait orientation, UI:276–277
scaling and resizing, UI:268–273
scrollbar, AR1:453
scrollwin, AR1:460–461
speedup, AR1:365
tab bars, AR1:508
table, AR1:383
constraints, AR1:386–387
flags, AR1:384–385
specifying, AR1:385–386
structure, AR1:384–385
using tAlignBaseline for, AR1:388
toolkit table, AR1:437
units, UI:279; AR1:375
vertical, UI:182
window, AR1:216, 224–225, 247–248
adding child windows to, AR1:381
classes, AR1:381–382
dirty, AR1:249, 365
episodic, AR1:225
parent-veto, AR1:225
processing, AR1:249
shrink-wrap and, AR1:397
unconstrained, AR1:224
windows, AWG:122
Layout classes, AR1:381–398; PDT:16
capturing vs. layout, AR1:398
coordinate system, AR1:382
custom layout, AR1:389–390
creating, AR1:390–395
layout loops, AR1:397–398
lazy layout, AR1:397
shrink-wrap, AR1:397
constraints and, AR1:395–396
table layout, AR1:383
window layout, AR1:381–382
Layout option sheet, UI:88
Connected Disks page, UI:111
Bookshelf view, UI:112
Directory view, UI:111
Connected Printers page, UI:117
Network View page (Disks section), UI:115
Show list, UI:88
lbFreeDataByMessage flag, AR1:466
Leaf-to-root model, AR1:565
Left Arrow gesture, UI:25
Left-Down gesture, UI:26
in gesture mode, UI:259
guidelines, UI:240
Left-Up gesture, UI:26
in gesture mode, UI:259
guidelines, UI:240
Legibility, AWG:77
Libraries, AWG:102
Life cycles, AR1:95–115
application class, AR1:95–99
deinstallation, AR1:99
installing, AR1:96–99
document, AR1:100–115
activating, AR1:102–107
closing, AR1:109–110
creating, AR1:102
deleting, AR1:115
opening, AR1:107–109
reactivating, AR1:113–115
terminating, AR1:110–113
Line
cap and join, AR1:278
deletion, AR2:269
drawing, AR1:337–338
end points in one-pixel, AR1:337
modes, AR1:291
number, PDT:88
patterns, AR1:271, 291–292
graphic state element, AR1:278
segments, AR1:265
smoothing, PDT:186
stoking and, AR1:290
styles, AR1:290–291
thickness, AR1:291
graphic state element, AR1:278
in rectangle, AR1:341
width, AR1:339–340
physical, AR1:340
lineCount, PDT:88
Line Height menu (MiniNote Paper sheet), UI:138
Line height preference, AR2:366
Link
files, AR1:134
protocols, AR2:253
Linker, AWG:93
flags, AWG:92–93
linking DLLs and, AWG:126
options, AWG:93
Watcom, AWG:126
Linking application, AWG:93
see also Compiling and linking
LIST_BOX_ENTRY structure, AR1:465
free mode, AR1:466
inserting and removing entries and,
AR1:467
setting state with, AR1:467
List boxes, AR1:463–473; PDT:16
contents of, AR1:464–465
creating, AR1:464
tables, AR1:465
inserting and removing, AR1:467
supplying, AR1:465–467
filing, AR1:469
font, AR1:473
gestures, AR1:468
messages, AR1:463–464, 469
modifying, AR1:467
notification, AR1:467–468
painting, AR1:468–469
pre-loading, AR1:467
scrolling, AR1:466
state, AR1:467
string, AR1:470–473
toolkit table vs., AR1:470
see also clsListBox; Lists
LIST_BOX_NEW structure, AR1:464
LIST_BOX_STYLE structure, AR1:464
Listener
field, AR1:549
objects, AR1:548, 549
LIST_ENTRY structure, AR2:129
LIST_ENUM structure, AR2:130
LIST_FREE structure, AR2:131
LIST_NEW structure, AWG:49; AR2:129
List object, AWG:45
attributes, AWG:47
code to create, AWG:51–52
messages and, AWG:46
see also Object
Lists, UI:29–32, 181–188
accessing end of, AR2:130
boxed, UI:32
checkboxes, UI:32
checklists, UI:29
with fields, UI:186
multiple, UI:31
pop-up, UI:30
vs. boxed, UI:186
class, AR2:127–131
concepts, AR2:127
creating, AR2:129
defined, AR2:127
destroying, AR2:131
discontiguous selection in, UI:282–283
editable, UI:187
items
adding, AR2:129–130
counting, AR2:130
enumerating, AR2:130–131
getting, AR2:129–130
removing, AR2:129–130
Logical unit coordinates (LUC), AR1:267–268, 335
coordinate systems transformations, AR1:268
drawing coordinates and, AR1:286–287
graphic state element, AR1:277
messages to set, AR1:287
obtaining, AR1:336
origin, AR1:292
scaling fonts and, AR1:276, 310
system, figure, AR1:335
transforming, AR1:268
units, AR1:287
unit size, AR1:268–269
Logical window coordinates (LWC), AR1:232, 267–268
LOGITECH, sample definition, PDT:62
Log Size menu (System Log), PDT:143
Low-level point events, AR1:10
Macintosh
file system, AR2:439
networking with, ARG:8
using S-Shot files on, PDT:177
Macros
debugging, PDT:67
DEBUG warning, AR1:23
declaration, AR1:39–40
error-checking, AR1:23–24
error handling, ARW:80, 81–84
expression handling, PDT:139
extensions of, ARW:76–84
for extracting information from UIDs, AR1:12
message passing, PDT:138
message sending, AR1:23–24
stack trace and, ARW:82
status checking, PDT:134
for testing UIDs, AR1:13
see also specific macros
Magnify Text on Screen control (MiniText), UI:132
Main entry point, ARW:22, 24
main function, ARW:71, 98; AR1:72
activating application and, ARW:107
in application installation, AR1:96–97
complex explanation of, ARW:106
in document activation, AR1:102–103
initialization routine and, ARW:100;
AR1:97
simple discussion of, ARW:105–106
Main notebook, ARW:33
Main routine, ARW:25, 98; AR1:33
calling, ARW:98
declaration, ARW:98
running process and, ARW:104
Main table of contents
Access Speed and, UI:87
Comments option sheet and, UI:87
option sheets and, UI:87
see also Table of contents
Main window, ARW:26; AR1:92
application, AR1:504
creating, AR1:504
inserting custom window as, AR1:561–562
in document activation, AR1:105
embedded applications and, ARW:35
initializing clsSPaper-based, AR1:593–594
setting, AR1:163
MakeDynUUID, AR1:80
MAKEFILE, ARW:91
for Counter Application,
ARW:203–204
for EmptyApp, ARW:179–180
for Hello World (custom window), ARW:194–195
for Hello World (toolkit), ARW:186
for Template Application,
ARW:106
for TTT (Tic-Tac-Toe),
ARW:249–250
Makefiles, ARW:92; AR2:403–404
Hello World (custom window),
ARW:127
Tic-Tac-Toe, ARW:164, 165
MakeMsg() macro, AR1:11
class UIDs and, AR1:568
MakeStatus() macro, ARW:80; AR1:11
MakeTag() macro, AR1:12, 235; AR2:167
defining window tags and, AR1:436
MakeWarning() macro, ARW:80; AR1:12
MakeWKNO() macro, ARW:53; AR1:10
MakeWknResld() macro, AR2:343;
PDT:173
MAKELABEL utility, PDT:162–163
every, PDT:163
Manager, AR1:438
button, AR1:438–439
choice, AR1:443
classes, AR1:423, 439–440
for menu buttons, AR1:166
objects, AR1:423
for toolkit tables, AR1:438
button manager notification
details, AR1:438–439
menu management, AR1:439
Manuals, ARW:175
Map files, PDT:148
Mapping, UI:290; AR1:130–131
of Adobe standard encoding to AFII
codes, PDT:200–204
file to memory, AR2:73–74
Memory Measurement, units of, AR1:131
Max macro, AR1:289
Margin rectangle region, AR1:379
Margins
gesture, UI:135–136
pop-up menu for, UI:58
Mark,
copying, AR1:203
creating, AR2:196
creating and holding, AR1:133
delivery messages, AR1:201–202
positioning, AR1:202
sent to components that have children, AR1:202
holder of, AR1:129
messages, AR1:132
parts, AR1:130
protocol, UI:171–172
Spell, Proof, Find commands and, UI:172
sending message to, AR1:202–203
setting, to component, AR1:203
supporting, AR1:132
MarkHandlerForClass() function, AR2:197
Marks class, AR1:69, 129–130, 199–203
embedded windows and, AR1:118
messages, AR1:199–200
see also clsMark
Markup layer, UI:253
using ink and, UI:255
Markup mode, UI:30–31
Marquee, in copy move operations, UI:68–69
Mask,
PDT:168
application, AWG:38
class, AR2:415–416
in node attribute flags, AR2:79
Paint mode, PDT:168
defined, PDT:171
text attribute messages, AR2:16
see also Icon mask
Math run-time library, AR2:115–117
programmatic interface, AR2:115–117
Matrix
manipulation messages, AR1:282
transformation, AR1:289
Max macro, AWG:78
Measurement, units of, AR2:10
Memory
Access Speed control and, UI:67
for application code, AR1:92
checking, AWG:112
available, PDT:81
conservation, AWG:15
in application termination, AWG:38–39
DB and, PDT:81–82
document state and, AWG:39
freeing, AR2:88, 95
globally accessible, PDT:129
instance data and, AR1:34
management, AR2:101–103
80386 protected mode, AR2:102
heaps, AR2:101–102
privilege levels, AR2:103
rings, AR2:103
mapping file to, AWG:144
map size, AR2:73
PenPoint programs in, AWG:23
protection, AR1:34
RAM, AR2:52
for running PenPoint, PDT:27
saving, PDT:47
multiple applications and, PDT:53
usage, UI:10
Usage (Show menu), PDT:142
using less, PDT:39
watching, PDT:82–84
Memory-mapped files, AWG:15, 89;
AR2:55
to avoid duplication, AWG:143
file system and, AWG:7
function, AR2:73
life cycle, AR2:73–74
sharing, AR2:73
Memory-resident file system, AWG:16
Memory-resident volumes, AR2:52
Menu bar, AWG:113; AR1:363
browser, AR2:138
in CounterApp, AWG:147
creating, AR1:353
MENU_BUTTON_NEW_ONLY structure, AR1:446
Menu buttons, AWG:113, 147–148;
UI:29; AR1:163–164, 418;
PDT:16
adding, AR1:166
creating, AR1:446–447
defining, AWG:170
displaying submenus with,
AR1:448–450
document and edit menus,
AR1:165–166
managers for, AR1:166
messages, AR1:445
painting, AR1:447
pop-up menu, AR1:445
specifications for, AWG:147
style of, AR1:446
tags, AR1:164
see also clsMenuButton
MENU_BUTTON_STYLE structure, AR1:446
Menu labels, UI:42
for current mode, UI:249
Menu line, UI:56
bold buttons on, UI:177
Connected Disks page, UI:107
designing, UI:226
hiding, UI:56
illustrated, UI:55
MiniNote, UI:136
MiniText, UI:129
for mode switch, UI:251
Network View page (Disks section), UI:114
table of contents, UI:85
zoom control in, UI:266–267
MENU_NEW_ONLY structure, AR1:447
Menus, UI:41–42, 192–199; AR1:428,
447–448; PDT:16
Back menu, PDT:173
behavior of, UI:42
checklists in, UI:184
closing, UI:42
Connected Disk page, UI:107
Connected Printers page, UI:116–117
controls in, UI:41–42
creating, AR1:447–448
submenus, AR1:448
deciding when to use, UI:226
default, UI:56
displaying, AR1:448
dividing lines in, UI:197
Document menu, PDT:171
Edit menu, PDT:171
FEDIT
Bitmap menu, PDT:192–195
Character menu, PDT:179, 182,
188
File menu, PDT:179–180, 196
Options menu, PDT:184–185, 191
Outline menu, PDT:183, 195
Text menu, PDT:198
fields in, UI:199
font face, UI:271
font size, UI:271
frame, AR1:503
hierarchical, UI:42, 199
Ink menu, PDT:173
Installed Software section, UI:98
layout of, UI:196–197
groups of controls separation and,
UI:197
multi-column menus, UI:196
management, AR1:439
for margins, UI:58
MiniNote, UI:136–137
Arrange menu, UI:137
Edit menu, UI:136–137
Pen menu, UI:137
MiniText, UI:129–130
default, UI:129
View menu, UI:130
mode switch following, UI:251
mode switch in, UI:251
nesting of controls in, AR1:363
for network disks, UI:114
network printers and, UI:119
Options menu, PDT:172–173
Out box, UI:122
Size menu, PDT:173
standard, UI:192–196
  Document menu, UI:192–194
  Edit menu, UI:192, 194–195
  Options menu, UI:192
  View menu, UI:195–196
  standard application, AR1:363
  strengths and weaknesses of, UI:225
submenus and, UI:42
summary, UI:40
System log application, PDT:142–143
Font menu, PDT:143
Log Size menu, PDT:143
Show menu, PDT:142
Trace menu, PDT:142–143
table of contents, UI:85–87
Create menu, UI:86–87
Document menu, UI:85
Edit menu, UI:86
Sort By menu, UI:88
View menu, UI:86
two-state switches in, UI:198
View menu (Connected Disk), UI:108–110
working of, AR1:363
with zoom control, UI:266–267
see also clsMenu; specific types of menus
Menu support, AWG:146–148

Merging
  shapes, PDT:189
  winding direction and, PDT:189–190
Message arguments, AWG:45–46; AR1:14
different expected, AWG:122
instance data and, AWG:138
msgNew, AWG:118–119
msgRestore, AWG:142
msgSave, AWG:141
new class, AWG:54–55
ObjectCall and, AWG:47
  as ObjectCall parameter, AWG:46
passing wrong, AWG:121
specific structure of, AWG:46
structure, AWG:47, 51; AR1:14–15
new object, AR1:15–17
Message handler, AWG:55, 109–111; AR1:18, 31
code, AR1:19
defined, AR1:5
designing, AWG:60
method table and, AWG:99
for msgDestroy, AWG:100
parameters, AWG:99, 109–110
  in EmptyAppDestroy, AWG:110
privacy, AWG:111
responses, AWG:57
status return values and, AWG:110
status values and, AR1:12
see also Methods
Message handling, AWG:107–109; PDT:14
method table and, AWG:108
msgDestroy, AWG:109
Message line, UI:214
Message parameters, AR1:38–39
Message passing
  functions, PDT:138
macros, PDT:138
  synchronous and asynchronous, PDT:14
Messages, AWG:41, 79; AR1:5
abstract, AWG:57
activation and deactivation, AR1:481
advisory, AWG:57
ancestor, AWG:60
auxiliary notebook manager, AR2:423
generalized, AR2:423–424
specialized, AR2:424–426
benefits of, AWG:43
boot error, PDT:47
boot progress, AR1:155
boot progress, AR2:431–432
clsAppDir, AR1:157–161
advanced, AR1:171
Application Framework and,
  AR1:161
class, AR1:157
document attributes, AR1:158
document hierarchy, AR1:158,
  161–163
document life cycle, AR1:157–158
document window, AR1:159, 163
observer, AR1:160
printing, AR1:160
standard application menu,
  AR1:159–160, 163–170
clsAppInstMgr, AR2:415
clsAppInstMgr, AR2:415
clsAppMonitor, AR1:152–153
  handling, AR1:155
  using, AR1:153–154
clsAppWin, AR1:195
clsBitmap, AR1:329–330
clsBorder, AR1:371–373
clsBrowser, AR2:138–140
for displayed information, AR2:142
  menu, AR2:145
  notification, AR2:145
  for sort order, AR2:142
clsButton, AR1:417–418
clsByteBuf, AR2:208
clsChoice, AR1:443
clsChoiceMgr, AR1:440
clsCodelnstallMgr, AR2:415
clsCodelnstallMgr, AR2:415
clsControl, AR1:401–402
class, AR2:409
instance, AR2:409–410
notification, AR2:408
subclass, AR2:410
using, AR2:409–413
clsOBXService, AR2:316
clsIP, AR1:585–586
clsLabel, AR1:410
clsList, AR1:218
functions, AR1:218
clsListBox, AR1:449, 463–464
clsMark, AR1:129, 199–200
clsMenu, AR1:447
clsMenuButton, AR1:445
clsMenuButton, AR1:445
clsNPData, AR2:234–235
clsNPData, AR2:234–235
clsNPData, AR2:234–235
clsNPData, AR2:234–235
clsNPData, AR2:234–235
clsNPData, AR2:234–235
clsNPData, AR2:234–235
clsNPData, AR2:234–235
clsNPData, AR2:234–235
clsNPData, AR2:234–235
clsNPData, AR2:234–235
class, AR2:157–158
class, AR1:128
class, AR1:317
  drawing, AR1:318
class, AR2:461–462
class, AR1:187
  using, AR2:187–188
clsResFile, AR2:347–348
clsRootContainerApp, AR1:187
clsScribble, AR1:608
class, AR1:154
in clsSelection, AR1:128
class, AR1:154
in clsSelection, AR1:128
class, AR1:154
in clsSelection, AR1:128
class, AR1:154
in clsSelection, AR1:128
clsSelection, AR1:154
clsSelection, AR1:154
clsSelection, AR1:154
clsSelection, AR1:154
clsService
  change ownership protocol,
    AR2:467–469
  information messages, AR2:459
  notification messages,
    AR2:461–462
  responsibility, AR2:469–470
class, AR1:140
class, AR1:140
class, AR1:140
clsSPaper, AR1:589–590
clsStream, AR2:133
clsString, AR2:212
clsStringListBox, AR1:470
clsSystem, AR2:431
clsTable, AR2:217–218
   information, AR2:226
clsTableLayout, AR1:384
clsTiff, AR1:331
clsTextIP, AR2:470
clsTable, AR2:217–218
clsUndo, AR2:202
clsWin, AR1:230–231
clsXferStream, AR2:171
class System, AR2:431
class StringListbox, AR1
class Stream, AR2:133
class SPaper, AR1:589-590
class Tiff, AR1:331
class TextIP, AR2:470
class Table, AR2:217-218
connection status, AR2:247
completion, AR1:5
coding conventions for, AWG:71
definition, AR1:5
delayed input, AR1:483
delivery, AR1:201–202
designing, AWG:60
drawing context, AR1:281–284
Empty Application, AWG:97
event, AR1:546–547
field component creation, AR1:480
file system, AR2:62–64
frame action, AR1:502
handling, AWG:44–45
header, AWG:75
help, UI:215
identifiers, AR1:11
input
   processing, AR1:482
   subsystem, AR1:570
   system, AR1:570
   validation, AR1:484
in instead of function calls, AWG:42–43
intercepted, AR1:127
intertask, AR2:100–101
   “Int w/o RB: 7,” PDT:47, 50
method table specifications for, AR1:41
move and copy protocol and, AR1:119
notification, AR1:399
object, AR1:49
   and class information, AR1:54
ObjectCall() and, AR1:19
ObjectPost() and, AR1:21–22
objects and, AWG:43
ObjectSend() and, AR1:20
observer, AR1:50, 150
Out box
protocol, AR2:308–309
response to, AR2:310
overriding, AR1:42
pattern specifications, PDT:117
possible responses to, AWG:57–58
preview, AR1:399, 405–406
message argument for, AR1:407
preview button, AR1:362
printing, AR1:135, 136
   protocol, AR1:140
progress, UI:211
progress bar, AR1:535–537
inherited, AR1:540
propagating, AR1:225
sampled images, AR1:297
search and replace, AR2:198
classes that respond to, AR2:198
to selection owners, AR1:195–196
self UIDs and, AWG:56–57
sendable services, AR2:333–334
sending, AWG:45–48; AR1:13–15
   Hello World (toolkit) and,
      AWG:115–116
      macros for, AR1:23–24
      message argument structure and,
         AR1:14–15
      methods of, AWG:115
      to object, AR1:15
      return values and, AR1:14
      sending with DB, PDT:82
      sent by holders, AR1:202–203
      sent by service managers,
         AR2:459–470
      sent to components, AR1:201–202
      sent to open services, AR2:470
      sent to service class, AR2:456–459
      status values and, AWG:47, 52
      string names for, PDT:82
      superclass, AR1:155
      system directory, AR2:432
      Template Application and,
         AR2:405–406
      Text insertion pad, AR2:33
      text data, AR2:12–13
      observer, AR2:21
      text insertion pad, AR2:33
      text view, AR2:23–24
      theBusyManager, AR2:193
      to theSelectionManager, AR2:161–162
      theTimer, AR2:104
      tracing, AWG:120, 159–160
      translation, AR1:604–606
      control, AR1:606
      initialization, AR1:605
      notification, AR1:606
types of, PDT:14
volume specific, AR2:91
window
   creation, AR1:230
   display, AR1:230–231, 235–244
   filing, AR1:231
   layout, AR1:231, 244–251
   management, AR1:231, 251–253
   metrics, AR1:230, 233–234
   sending to hierarchy, AR1:251–252
   wording of, UI:210
see also specific messages
Message status, AR1:18–19
Message table, AWG:56
Message text, AR1:495–496
message string format codes, AR1:495–496
specifying command buttons, AR1:495
Metaclasses, AR1:84–85
clsAppMgr and, AR1:85
Methods, AR1:31
ancestor calls and, AR1:36–37
code for, AR1:34
creating, AR1:37–41
declaration entry points, AR1:37
message parameters, AR1:38–39
operations and, AR1:41
handling search and replace functions, AR2:197
for msgAppClose, AR1:109
for msgAppInit, AR1:105
for msgAppOpen, AR1:108
for msgFree, AR1:110
for msgRestore, AR1:105, 114
for msgSave, AR1:105, 112
see also Message handler
METHODS.TBL, AWG:91, 99
Counter Application, AWG:196–197
Empty Application sample code, AWG:178
Hello World (toolkit), AWG:182
Template Application, AWG:252
TrtApp, AWG:206–207
Method tables, AWG:47, 49, 55–58;
   AR1:31
class implementation and, AR1:32
CLASS_NEW message argument, AWG:54
for clsCntt, AWG:138
for clsCnttApp, AWG:137–138
for clsHelloWin, AWG:125, 127
for clsHelloWorld, AWG:125, 127
for clsList, AWG:57–58
compiler, AWG:55, 56, 92
   header file and, AWG:109
   see also MT (method table compiler)
compiling, AWG:93; AR1:45–46
two steps, AR1:46
with wildcards, AR1:46
creating, AR1:41–46
   build sequence, AR1:43
   five steps for, AR1:42
   definition file, AR1:41
   creating, AR1:42–45
in document activation, AR1:105
Empty Application and, AWG:91
entry for msgAppOpen, AR1:107
files, AWG:55, 98
   creating, AWG:66
   suffix, AWG:98
for Hello World (toolkit), AWG:114
message handlers and, AWG:99
names, AWG:98
overview, AR1:41–42
wild cards, AWG:98
wildcards, AR1:45
Metrics, AWG:120; AR1:145
ASCII, transfer, AR2:175
bitmap, AR1:130
browser, AR2:143–144
caller, AR1:311
for controls, AWG:123
embedded window object, AR2:161
for controls, AWG:123
font, AR1:275,
embedded window object, AR2:161
for controls, AWG: 123
character,
getting, for class, AR1: 149
getting, embedded window,
getting, application window,
getting, for class, AR1:149
NotePaper, AR2:230
setting format, PDT:29
file list of,
description, PDT:28
in boot sequence, PDT:29
keywords, PDT:33-34
in boot sequence, PDT:29
mono debug line, PDT:137
setting format, PDT:29
single monitor and, PDT:44
UNIPENPORT tag, PDT:60
UNIPENPROTOCOL tag, PDT:61
UNIPENTYPE tag, PDT:60
for writing to serial port, PDT:136
MIL (Machine Interface Layer), UI:5;
AWG:6; AR2:98; PDT:29
during booting, PDT:46
for PC, PDT:29
Service, AWG:273–274
tablet hardware, PDT:29
see also MIL services
MIL services, AR2:98, 439
binding, AR2:247
connection management,
AR2:247–248
connection status messages, AR2:247
connectivity, AR2:245–246
other services and, AR2:246–249
functions, AR2:245
ports and, AR2:246
programmatic interface, AR2:246
stream interface and, AR2:246
see also Parallel port; Serial port
MILSVC service, AR2:487–506
defined, AR2:475
METHOD.TBL, AR2:487–489
MILSVCC.C, AR2:489–499
MILSVCO.C, AR2:500–506
MILSVCO.H, AR2:499–500
mini command, PDT:101
Mini-debugger, AWG:84, 85;
PDT:145–154
for code crashes, AWG:87
commands, PDT:146–148
setting debug flags and,
PDT:147–148
exception handling, PDT:148
functions, PDT:145
invoking, PDT:145–146
mini-debugger and DB, PDT:145
on PenPoint computer, PDT:146
map files and, PDT:148
source level debugger (DB) and,
PDT:145
task list, PDT:151–154
understanding interrupts and,
PDT:148–151
uses of, AWG:111
using, PDT:148
see also Debugging
MiniNote, UI:135–142
document pagination, AR1:137
edit pads and, UI:47
gesture margin, UI:135–136
gesture mode, UI:135
gestures, UI:140–142, 257
capital letter accelerators, UI:142
gesture mode, UI:141–142
ink and gesture modes, UI:140
icon, UI:77, 216
ink mode, UI:135
menu line, UI:136
menus, UI:136–137
Arrange menu, UI:137
Edit menu, UI:136–137
Pen menu, UI:137
mode toggle, UI:135
option sheets, UI:138–139
Paper, UI:138–139
Pen, UI:139
reference buttons and, UI:171
MiniText, AWG:116; UI:129–134;
AR2:394
check and, UI:46
Document menu buttons, AR1:363
document pagination, AR1:137
etit pads and, UI:47
embedded writing pads, UI:49
file import/export, AR2:253
gestures, UI:133–134
icon, UI:77, 216
Insert menu, UI:195
menus, UI:129–130
option sheets, UI:130, 132, 204
Character, UI:131
correlation to objects, UI:204
Display, UI:132
Paragraph, UI:131
Tab Stops, UI:132
Options menu, UI:45
reference buttons, UI:171
labels, UI:172
text component and, UI:130
token implementation, AR1:131
writing pad, UI:48
Min macro, AWG:78
MISC directory, AR2:387, 394–395
data examples, AR2:395
service directory and, AR2:387, 444, 445
Miscellaneous application files, AR2:387
Mitosis operator, PDT:186
MKS toolkit, AWG:162
MM, sample definition, PDT:61
MNP mode, modem, AR2:286
data communication, AR2:289–290
Modal applications, UI:253
deferred and immediate translation,
UI:253
markup layer, UI:253
Modal dialog sheets, UI:165, 205–206
uses of, UI:206
Modal notes, UI:52
Modal objects, UI:43
distinction of, UI:43
Modeless dialog sheets, UI:205
advantages of, UI:205
Models, user, UI:245–246
for input modes, UI:246
selection and, UI:278
Modem interface, AR2:253
service, AR2:439
see also Data modem
MODEM_AUTO_ANSWER_SET structure,
AR2:284
MODEM_DIAL structure, AR2:287
MODEM_MNP_BREAK_TYPE_SET structure, AR2:290
MODEM_MNP_FLOW_CONTROL_SET structure, AR2:290
MODEM_MNP_MODE_SET structure,
AR2:286
MODEM_NEW_ONLY structure, AR2:283
MODEM_SEND_COMMAND structure,
AR2:287
Move protocol, AR:119–123
data type determination, AR:121
destination by user, AR:120
destination in file system, AR:122
destination to move, AR:121
getting exact pen location, AR:123
identifying selection, AR:120
moving data, AR:122–123
OK move, AR:121–122
reasons for using, AR:119
requesting move, AR:120

Moving
as adjusting, UI:289
between applications, AR:120
beginning, operation, AR:160–161
control points, PDT:185–186
and copying guidelines, UI:285–294
data, embedded window,
 AR:246–248

data, steps for, AR:119–120
docket & drop gesture for, UI:68
embedded windows, AR:126–128

between, AR:118–119
figure to far destination, UI:294
import/export and, UI:70
item in list, UI:292
in mouse-based interfaces, UI:285

nodes, AR:280–81
in PenPoint, UI:286
picture segments, AR:326–327
in Tic-Tac-Toe, AR:123–127
variations, UI:289–290
windows, AR:246–247
word in text, UI:69, 291

see also Copying

MS-DOS
C development tools, PDT:6
disk drive class, AR:247
FAT disk format, AR:251
creating PENPOINT.DIR and,
 AR:268

file system, AR:439
compatibility, AWG:7–8
file system utilities, PDT:161–164
GDIR, PDT:162
MAKELABEL, PDT:162–163
PAPPEND, PDT:163–164
PDEL, PDT:164
PSYNC, PDT:164
STAMP, PDT:161–162

installation, AWG:21–22
LABEL command, PDT:72
main routine, AWG:25
name, PDT:162
networking with, AWG:8
termination, AWG:23
tools, PDT:13
volume name, AR:251; PDT:72

msgABMgrActivate, AR:2:329
status values, AR:2329–330
msgABMgrChanged, AR:2:330
msgABMgrClose, AR:2:326
msgABMgrDelete, AR:2:326
msgABMgrOpen, AR:2:326
msgABMgrRegister, AR:2:329
msgABMgrUnregister, AR:2:329
msgActivateChild, AWG:107
msgAdded, AR:1:51

translation and, AR:1:605

msgAddObserver, AR:1:27, 50–51; AR:2:49
observing installation managers and,
 AR:2:407

observing tables and, AR:2:220
system preferences and, AR:2:368
msgAddObserverAt, AR:1:50–51
msgAddrBookAdd, AR:2:328
msgAddrBookDelete, AR:2:328
msgAddrBookGet, AR:2:327–328
msgAddrBookSearch, AR:2:326, 327
msgAddrBookSet, AR:2:328
msgAMGetMetrics, AR:1:153
msgAMLoadHelp, AR:1:154
msgAMLoadMisc, AR:1:154
msgAMLoadStationary, AR:1:153
msgAMRemoveHelp, AR:1:154
msgAMRemoveStationary, AR:1:154
msgAMTerminateOK, AR:1:155
msgAncestor, AR:1:55
msgAncestorsIA, AR:1:55
msgANMAddToStationaryMenu,
AR:2:426

msgANMCopyInDoc, AR:2:425
msgANMCreateDoc, AR:2:425
msgANMCreateSect, AR:2:424
msgANMDelete, AR:2:426
msgANMDeleteAll, AR:2:426
msgANMGetNotebookPath, AR:2:423
msgANMOpenNotebook, AR:2:423
msgANMRemoveFromStationaryMenu,
AR:2:427
msgANMSysInited, AR:2:423
msgAppActivate, AR:1:105–107
in document reactivation, AR:1:113

msgAppAddCards, AR:1:138, 168–169
handling, AR:1:138–139
responding to, AR:1:169
msgAppActivateFloatingWin, AR:1:163
managing multiple windows and,
 AR:1:500
msgBrowserSetClient, AR2:144
msgBrowserSetMetrics, AR2:143–144
msgBrowserSetSaveFile, AR2:143
msgBrowserSetSelection, AR2:141
msgBrowserShow messages, AR2:142
setting metrics and, AR2:144
msgBrowserUserColumnQueryState, AR2:146
msgBusyDisplay, AR2:143
msgBusySetXY, AR2:144
msgBusySetSelection, AR2:141
msgBusyWriteState, AR2:143
msgByteBufChanged, AR2:209
msgByteBufGetBuf, AR2:208
msgBusyNotifyManager, AR2:424
msgBusySetBuffer, AR2:208
msgBusyNotifyClient, AR2:208
msgBusyNotify, AR2:421
msgByteBufGetBuf, AR2:208

for incrementing value, AWG:140
msgBusyNotifyClient, AR2:208
msgBusyNotify, AR2:421

for incrementing value, AWG:140
msgBusyNotifyClient, AR2:208
msgBusyNotify, AR2:421

for incrementing value, AWG:140
msgBusyNotifyClient, AR2:208
msgBusyNotify, AR2:421
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>msgFSSetTarget</td>
<td>AR2:59, 67, 76 call example, AR2:86 in changing target directory, AR2:86 directory position and, AR2:88</td>
</tr>
<tr>
<td>msgFSvolName</td>
<td>AR2:91</td>
</tr>
<tr>
<td>msgFSTraverse</td>
<td>AR2:67, 81–82 in call back routine, AR2:82 function of, AR2:81</td>
</tr>
<tr>
<td>msgFSvolSpecific</td>
<td>AR2:91</td>
</tr>
<tr>
<td>msgGetInstalledVolumes</td>
<td>AR2:49</td>
</tr>
<tr>
<td>msgGetObserver</td>
<td>AR1:52</td>
</tr>
<tr>
<td>msgGetPath</td>
<td>AR2:85 example of, AR2:86</td>
</tr>
<tr>
<td>msgGWinForwardedGesture</td>
<td>AR1:369, 408 gesture propagation and, AR1:369 gesture windows and, AR1:368 responding to gestures and, AR1:369</td>
</tr>
<tr>
<td>msgGWinGesture</td>
<td>AWG:156; AR1:127 application response to, AR1:170 gesture propagation and, AR1:369 gesture windows and, AR1:368 responding to gestures and, AR1:369</td>
</tr>
<tr>
<td>msgGWinTransformGesture</td>
<td>AR1:369</td>
</tr>
<tr>
<td>MsgHandlerArgType()</td>
<td>AR1:34, 98 using, AR1:40</td>
</tr>
<tr>
<td>MsgHandler()</td>
<td>AR1:39</td>
</tr>
<tr>
<td>MsgHandler() naming pointer, AWG:132</td>
<td></td>
</tr>
<tr>
<td>MsgHandler() using, AR1:39</td>
<td></td>
</tr>
<tr>
<td>MsgHandlerParameterNoWarning()</td>
<td>AR1:39 using, AR1:40</td>
</tr>
<tr>
<td>MsgBox</td>
<td>AR1:110</td>
</tr>
<tr>
<td>MSG_HANDLER structure</td>
<td>AR1:37</td>
</tr>
<tr>
<td>MsgHandlerWithTypes()</td>
<td>AWG:99, 132; AR1:39 using, AR1:40</td>
</tr>
<tr>
<td>MsgHandler() macro</td>
<td>AR1:39</td>
</tr>
<tr>
<td>MsgIconCopyPixels</td>
<td>AR1:525</td>
</tr>
<tr>
<td>MsgIconFreeCache</td>
<td>AR1:525</td>
</tr>
<tr>
<td>MsgIconProvideBitmap</td>
<td>AR1:524 in cached picture and, AR1:525</td>
</tr>
<tr>
<td>MsgIMCurrentChanged</td>
<td>AR2:408</td>
</tr>
<tr>
<td>MsgIMDeinstalled</td>
<td>AR2:256, 409 in deinstalling service, AR2:465 in deleting installable item, AR2:412</td>
</tr>
<tr>
<td>MsgIMDelete</td>
<td>AR2:412</td>
</tr>
<tr>
<td>MsgIMDup</td>
<td>AR2:412</td>
</tr>
<tr>
<td>MsgIMGetCurrent</td>
<td>AR2:412</td>
</tr>
<tr>
<td>MsgIMGetDir</td>
<td>AR2:411</td>
</tr>
<tr>
<td>MsgIMGetList</td>
<td>AR2:261, 413, 417</td>
</tr>
<tr>
<td>MsgIMGetName</td>
<td>AR2:258, 262, 417</td>
</tr>
<tr>
<td>MsgIMGetNotify</td>
<td>AR2:407</td>
</tr>
<tr>
<td>MsgIMGetSema</td>
<td>AR2:407, 414</td>
</tr>
<tr>
<td>MsgIMGetSize</td>
<td>AR2:413</td>
</tr>
<tr>
<td>MsgIMGetState</td>
<td>AR2:412, 413</td>
</tr>
<tr>
<td>MsgIMGetStyle</td>
<td>AR2:411</td>
</tr>
<tr>
<td>MsgIMInstall</td>
<td>AR2:254, 411</td>
</tr>
<tr>
<td>MsgIMInstalled</td>
<td>AR2:409</td>
</tr>
<tr>
<td>MsgIMInUseChanged</td>
<td>AR2:408</td>
</tr>
<tr>
<td>MsgIMModifiedChanged</td>
<td>AR2:408</td>
</tr>
<tr>
<td>MsgIMNameChanged</td>
<td>AR2:408</td>
</tr>
<tr>
<td>MsgImport</td>
<td>AR2:151–152</td>
</tr>
<tr>
<td>MsgImportQuery</td>
<td>AR2:148 responding to, AR2:150–151</td>
</tr>
<tr>
<td>MsgIMSetCurrent</td>
<td>AR2:412</td>
</tr>
<tr>
<td>MsgIMSetModified</td>
<td>AR2:407</td>
</tr>
<tr>
<td>MsgIMSetName</td>
<td>AR2:412</td>
</tr>
<tr>
<td>MsgIMSetNotify</td>
<td>AR2:407</td>
</tr>
<tr>
<td>MsgIMSetStyle</td>
<td>AR2:411</td>
</tr>
<tr>
<td>MsgINBXSvcPollDocuments</td>
<td>AR2:313</td>
</tr>
<tr>
<td>MSG_INFO array</td>
<td>AR1:43–44 for clsTTdTData, AR1:44 defined, AR1:42 entry fields, AR1:44 option flags, AR1:45 wildcards and, AR1:45</td>
</tr>
<tr>
<td>msgInputEvent</td>
<td>AWG:156; AR1:546 input event status codes and, AR1:569 msgInputGrabPushed, AR1:572 msgInputGrabTerminated, AR1:549 msgInsertSibling, AR1:233</td>
</tr>
<tr>
<td>msgIPDataAvailable</td>
<td>AR1:588</td>
</tr>
<tr>
<td>msgIPGetXlateData</td>
<td>AR1:588</td>
</tr>
<tr>
<td>msgIPSetTranslator</td>
<td>AR1:598</td>
</tr>
<tr>
<td>msgIsA</td>
<td>AR1:54, 367 failures during, AWG:153; AR1:59 method for, AR1:104</td>
</tr>
<tr>
<td>msgInputEvent</td>
<td>AR1:546</td>
</tr>
<tr>
<td>msgInputGrabPushed</td>
<td>AR1:572</td>
</tr>
<tr>
<td>msgInputGrabTerminated</td>
<td>AR1:549</td>
</tr>
<tr>
<td>msgInsertSibling</td>
<td>AR1:233</td>
</tr>
<tr>
<td>msgKeyChar</td>
<td>AR1:583 event data, AR1:583 msgKeyMulti and, AR1:583</td>
</tr>
<tr>
<td>msgKeyDown</td>
<td>AR1:582 event data, AR1:582 msgKeyMulti and, AR1:583</td>
</tr>
<tr>
<td>msgKeyMulti</td>
<td>AR1:583–584 event data, AR1:584</td>
</tr>
<tr>
<td>msgKeyUp</td>
<td>AR1:582 event data, AR1:582</td>
</tr>
<tr>
<td>msgLabelGetRects</td>
<td>AR1:416</td>
</tr>
<tr>
<td>msgLabelGetString</td>
<td>AR1:486</td>
</tr>
</tbody>
</table>
creating objects and, AR1:34
creating progress bar and, AR1:535
creating resource file handle and,
AR2:348
creating resource list and, AR2:346
creating scribble object and, AR1:609
creating service instances and, AR2:454
creating stream object and, AR2:134
creating string object and, AR2:212
creating table object and, AR2:220
creating text data objects and, AR2:7, 13
creating text insertion pad and, AR2:33
creating text view object and, AR2:5, 24
creating toolkit tables and, AR1:435
creating translator and,
creating toolkit tables and, AR1:55
for initialization, AWG:119
in opening file, AR2:46
UI Toolkit classes and, AR1:364
using, AR1:17
window style flags and, AR1:232
msgNewWithDefaults, AR1:18
msgNoteCancel, AR1:492
msgNoteDone, AR1:491
msgNotifyShow, AR1:490
msgNotifyObservers, AR1:27, 52–53
msgNotUnderstood, AR2:159
MsgNum, AWG:156
msgNumObservers, AR1:52
msgObjectClass, AR1:55
msgObjectIsA, AR1:54
msgObjectNew, AR1:128; AR2:456
msgObjectOwner, AR1:55
msgObjectValid, AR1:55
msgObjectVersion, AR1:56
msgOBXDocOutputDone, AR2:309
msgOBXDocLockDocument, AR2:309
msgOBXDocUnlockDocument, AR2:309, 310
msgOBXDocMoveInDoc, AR2:309
msgOBXDocNextDocument, AR2:309
msgOBXDocOutputStart, AR2:311
msgOBXDocPollDocuments, AR2:309
msgOBXDocUnlockDocument, AR2:309, 310
msgOpen, AWG:36
clsHelloWorld and, AWG:127
msgOptionAddCard, AR1:138, 515; AR2:318
msgOptionAddLastCard, AR1:139, 169
msgOptionApplicableCard, AR1:516, 518–520
msgOptionApply, AR1:516
msgOptionApplyCard, AR1:516, 520
msgOptionClose, AR1:516
msgOptionCreateSheet, AR1:519
msgOptionDirtyCard, AR1:520
msgOptionExtractCard, AR1:516
msgOptionProvideCard, AR1:516
msgOptionRefreshCard, AR1:516
msgOptionRemoveCard, AR1:515
msgOptionSetCard, AR1:515
msgOptionSheetAddCards, AR2:330
msgOptionShowCard, AR1:515
msgOptionSetCard, AR1:516
msgObjectValid, AR1:55
msgObjectOwner, AR1:55
msg parameter, AWG:99, 109; AR1:38
msgOwner, AR1:55
msgOutProxUp, AR1:575
msgPenDown, AR1:407, 563, 575
event data, AR1:576
msgPenDown, AR1:407, 563, 575
event data, AR1:576
msgPenEnterDown, AR1:575, 577
event data, AR1:577
msgPenExitDown, AR1:575, 577
event data, AR1:578
msgPenExitUp, AR1:575, 578
event data, AR1:578
msgPenHoldTimeout, AR1:120
msgPenInProxUp, AR1:575, 578
event data, AR1:578
msgPenMoveDown, AR1:575, 577
event data, AR1:577
msgPenMoveUp, AR1:575, 577
event data, AR1:577
msgPenOutProxUp, AR1:578–579
event data, AR1:579
msgPenStroke, AR1:579
event data, AR1:579
msgPenTap, AR1:579–580
event data, AR1:580
msgPenUp, AR1:407, 575, 576
event data, AR1:576
msgPicSegAddGrafic, AR1:325
msgPicSegChangeOrder, AR1:325
msgPicSegDelete, AR1:324
msgPicSegDelta, AR1:324
msgPicSegDrawGrafic, AR1:324
msgPicSegDrawGraficIndex, AR1:324
msgPicSegDrawObject, AR1:320, 323
msgPicSegDrawSpline, AR1:320, 323
msgPicSegErase, AR1:322
msgPicSegGetCount, AR1:324
msgPicSegGetGrafic, AR1:324
msgPicSegHitTest, AR1:325
msgPicSegMakeInvisible, AR1:326
msgPicSegPaint, AR1:319
msgPicSegPaintObject, AR1:323
msgPicSegRedraw, AR1:333
msgPicSegScaleUnits, AR1:327
msgPicSegSetCurrent, AR1:324
msgPicSegSetFlags, AR1:323
msgPopupChoiceGetChoice, AR1:451
msgPostObservers, AR1:53
msgPortAutoLineFeedOn/Off, AR2:277
msgPortCancelPrint, AR2:278
msgPortGetTimeDelays, AR2:277
msgPortInitialize, AR2:277–278
msgPortSetTimeDelays, AR2:277, 278
msgPortStatus, AR2:278
msgPrefsPreferenceChanged, AR1:170; AR2:368
msgPrintEmbeddeeAction, AR1:143
msgPrintGetProtocols, AR1:137, 140
not understood, AR1:143
msgPrintLayoutPage, AR1:141, 142
not understood, AR1:143
msgPrintStartPage, AR1:140
not understood, AR1:143
msgProgressGetFilled, AR1:538
msgProgressGetMetrics, AR1:536, 537
msgProgressGetStyle, AR1:536
msgProgressGetUnfilled, AR1:538
msgProgressGetVisInfo, AR1:538–539
msgProgressProvideLabel, AR1:539–540
msgProgressSetFilled, AR1:538
msgProgressSetMetrics, AR1:537
msgProp, AR1:28, 57
msgQuickHelpOpen, AR2:188
msgQuickHelpShow, AR1:370; AR2:181
in displaying Quick Help text,
AR2:187
msgRAAppGotoDoc, AR1:184, 187
msgRAAppNextDoc, AR1:187
msgRAAppPrevDoc, AR1:187
msgRCAppReferenceContents, AR1:188
msgRCAppPrevDoc, AR1:187
msgRCAppNextDoc, AR1:187
msgQuickHelpShow, AR1:370; AR2:181
in displaying Quick Help text,
AR2:187
msgRAAppGotoDoc, AR1:184, 187
msgRAAppNextDoc, AR1:187
msgRAAppPrevDoc, AR1:187
msgRCAppReferenceContents, AR1:188
msgRAAppCancelGotoDoc, AR1:188
msgRemoveObserver, AR1:27, 51; AR2:220
msgResAgent, AR2:353–354
msgResCompact, AR2:352, 353
msgResCreateResource, AR2:352
msgResEnumResources, AR2:351–352
msgResFindResource, AR1:517
stateful objects and, AWG:65
terminating document and, AR1:111, 112–113
in Tic-Tac-Toe application, AR1:113
writing objects/data and, AR2:44
msgScroll, AR2:127
msgScrAddedStroke, AR1:606
msgScrCat, AR1:609
msgScrClear, AR1:609
msgScrComplete, AR1:610
msgScrCompleted, AR1:606
msgScrCount, AR1:609
msgScrDeleteStroke, AR1:609
msgScrGetBounds, AR1:609
msgScrHit, AR1:610
msgScrollbarHorizScroll, AR1:454
normalizing scroll and, AR1:456
scroll windows and, AR1:457
msgScrollbarProvideHorizInfo, AR1:454
scroll windows and, AR1:457
msgScrollbarProvideVertInfo, AR1:454
normalizing scroll and, AR1:457
scroll windows and, AR1:457
msgScrollbarUpdate, AR1:456
msgScrollbarVertScroll, AR1:454
normalizing scroll and, AR1:456–457
scroll windows and, AR1:457
msgScrollWinAddClientWin, AR1:462
msgScrollWinCheckScrollbars, AR1:460
msgScrollWinGetInnerWin, AR1:457, 460
msgScrollWinGetStyle, AR1:460
msgScrollWinProvideDelta, AR1:461, 462
msgScrollWinShowClientWin, AR1:462
msgScrRemovedStroke, AR1:606
msgScrRender, AR1:610
msgScrStrokePtr, AR1:610
msgScrXtractComplete, AR1:606
msgSelBeginCopy, AR1:120; AR2:159, 160
msgSelBeginMove, AR1:120, 123–124; AR2:159, 160
msgSelChangedOwners, AR2:162, 163
restoring selection owners and, AR2:162
msgSelCopySelection, AR1:121
handling, AR1:128
picture segments and, AR1:327
msgSelDemote, AR2:159
handling, AR2:160
msgSelDelete, AR2:159
handling, AR2:160
preserving owner selection and, AR2:162
msgSelIsSelected, AR2:159
msgSelMarkSelection, AR1:128
msgSelMoveSelection, AR1:121, 125
handling, AR1:128
picture segments and, AR1:327
msgSelOptions, AR1:520, 521; AR2:159
handling, AR2:160
msgSelOptionTagOK, AR2:160
msgSelOwner, AR2:31
finding selection owners and, AR2:161–162
selection transitions and, AR2:157
msgSelPromote, AR1:521; AR2:159
handling, AR2:160
restoring selection owner and, AR2:162
msgSelPromotedOwner, AR2:163
msgSelSelect, AR2:161
msgSelSetOwner, AR2:157, 159, 162
clsEmbeddedWin and, AR2:161
promoting/demoting and, AR2:160
msgSelSetOwnerPreserve, AR1:521;
AR2:159, 162
c1sEmbeddedWin and, AR2:161
restoring selection owner and, AR2:162
msgSelYield, AR2:157, 159
handling, AR2:160
restoring selection owner and, AR2:162
msgSendServCreateAddrWin, AR2:332, 333
msgSendServFillAddrWin, AR2:332, 333–334
msgSendServGetAddrDesc, AR2:332, 333
msgSendServGetAddrSummary, AR2:332, 334
msgSetAttr, AR2:66
msgSetOwner, AR1:26
msgSetProp, AR1:28, 56–57
msgSetTranslator, AR1:588
msgShowMenu, AR1:448
msgSIMGetMetrics, AR2:456
msgSioBaudSet, AR2:269
data modem and, AR2:280
msgSioBreakSend, AR2:272
msgSioBreakStatus, AR2:273
msgSioControlInStatus, AR2:270
msgSioControlOutSet, AR2:270
msgSioEventGet, AR2:273
msgSioEventHappened, AR2:266, 272
msgSioEventSet, AR2:266, 272
msgSioEventStatus, AR2:273
msgSioFlowControlCharSet, AR2:270
msgSioFlowControlSet, AR2:269
msgSioGetMetrics, AR2:269–271
msgSioInputBufferFlush, AR2:271
msgTBLSemaClear, AR2:222
msgTBLSemaRequest, AR2:217, 222
msgTextAffected, AR2:21
msgTextChangeAttrs, AR2:19, 35

msgUndoCurrent, AR2:206
msgUndoEnd,AR2:203, 204

msgTextChangeCount, AR2:37
msgTextClearAttrs, AR2:19, 20
msgTextEmbedObject, AR2:20
msgTextEnumEmbeddedObjects, AR2:20
msgT extExtractObject, AR2:20

msgUndoGetMerrics, AR2:205
msgUndoltem, AR2:206

msgTextGet, AR2:14
msgTextGetAttrs, AR2:18, 35
msgTextGetBuffer, AR2:14
msgTextlnitAttrs, AR2: 19
msgTextLength, AR2: 14
msgTextModifY, AR2:15
counting changes and, AR2:37
msgTextReplaced, AR2:21
msgTextSpan, AR2:15
msgTextViewAddIP, AR2:26-27, 33
circle-line gesture and, AR2:26
overriding behavior of, AR2:27
msgTextViewCheck, AR2:32
msgTextViewEmbed, AR2:26
msgTextViewGetStyle, AR2:32
msgTextViewResolveXY, AR2:27-28
msgTextViewScroll, AR2:29
msgTextViewSetStyle, AR2:32
msgTiffGetMerrics, ARI :332
msgTimerAlarmNotifY, AR2:104
msgTimeRegister, ARl:408
msgTimerNotifY, AR2:104
msgTkTableAddAsFirst, ARI :436
msgTkTableAddAsLast, ARl:436
msgTkTableAddAsSibling, ARI :436
msgTkTableAddAt, ARl:436
msgTkTableGetManager, ARl:443
msgTkTableRemove, ARI :441
msgTPBind, AR2:301
msgTPRecvFrom, AR2:300
msgTPSendRecvTo, AR2:300
msgTPSendTo, AR2:299, 300
msgTrace, AWG:159, 160
msgTrackDone, ARl:528
msgTrackHide, ARl:528
msgTrackShow, ARl:528
msgTrackUpdate, ARl:528
msgTttDataChanged, AWG: 155
msgUndoAbort, AR2:204-205
msgUndoAddltem, AR2:203-204
to aborting transaction, AR2:204-205
msgUndoBegin, AR2:202-203

msgUndoFreeltem, AR2:206
msgUndoFreeltemData, AR2:203

msgUndoLimit, AR2:205
msgVersion, ARl:56
msgViewGetDataObject, ARl: 174;
AR2:26
msgViewSetDataObject, ARl: 174;
AR2:26
msgWin*OK messages, ARl:247
msgWinBeginPaint, ARI :240
picture segments and, ARl:319
msgWinBeginRepaint, AWG: 152;
ARl:238
painting with, ARI :239
picture segments and, ARl:319
scrollwin and, ARl:460
update region and, ARI :240
msgWinCleanRect, ARl:260
msgWinCopyRect, ARl:243
for scrollbar display and, ARl:457
msgWinDelta, ARl :234
laying out self and, ARl:250
resizing/moving and, ARl:246
scrollwin notification and, ARl:461
window layout and, ARl :382
msgWinDevBindPixelmap, ARl:257
accessing memory and, ARl:258
allocating pixehnap and, ARl:258
for multiple pixelmaps, ARl:262 .
msgWinDevGetRootWindow, ARI :255,
256
msgWinDirtyRect, ARl:239, 294
writing code and, ARI :241
msgWinDumpTree, ARI :252
msgWinEndPaint, ARl:240
msgWinEndRepaint, ARl:240
msgWinEnum, ARl :367
msgWinExtract, ARI :234
deleting insertion pad and, ARl:587
msgWinFindTag, ARl:235, 368
clsTkTable and, ARl:435
modifYing items in toolkit table and,
ARl:436
msgWinGetBaseline, ARl:251, 387
msgWinGetDesiredSize, ARl:249, 250
custom layout constraints and,
ARl:392
UI Toolkit and, ARl:368
msgWinGetFlags, ARl:235
.window input flags and, ARl:569
msgWinGetMerrics, ARl:233

msgWinGetTag, ARl:235
msgWinInsert, ARl:107, 212, 233-234
altering child windows and, ARl:246
insertion pad
creating, ARl:586
displaying, ARl:587
msgWinInsertSibling, ARl:228, 233
msgWinLayout, ARl:141, 247, 248
baseline alignment, ARl :251
caching desired sizes, ARI :250
client interface to layout and, ARl:249
function, ARl :381
laying out self, ARl:250
layout and geometry capture, AR1:251
layout episode, ARl:249
layout loop and, ARl :397
layout processing, ARl:249
in layout speedup, ARl:365
lazy layout and, ARl:397
progress bar and, ARl:536, 537
scrollwin layout and, ARl:460
selfs desired size, ARl:250
shrink-to-fit, ARI :250
UI Toolkit programming and,
ARl:366
when sent, ARl :248
window border layout and, ARl:378
msgWinLayoutSelf, ARl:249, 250, 381
clsCustomLayout and, ARI :382
clsTableLayout and, ARI :382
function, ARl:381
label child windows and, ARI :415
label layout and, ARI :414
scrollwin layout and, ARl :460
table layout consrraints and, ARl:386
UI Toolkit and, ARl:368
msgWinMoved, ARl:246-247
msgWinRepaint, AWG:128, 129, 187;
ARl:226,237
application printing and, ARI :302
child windows and, ARl:415, 416
explicitly painting and, ARl:240
image devices dirty windows and,
ARl:259
painting window with, ARI :239
repainting process and, ARl:242-243
repainting srrategy and, AWG: 152
window objects and, AWG:133
in writing paintlrepaint code, ARl:241
wsSynchRepaint flag and, ARl:242
msgWinSend, ARl:127, 368
in gesture propagation, ARl:369
toolkit table manager notification
and, ARl:438-439
toolkit table notification messages
and, ARI :438
window hierarchy and, ARI :251-252
msgWinSetFlags, ARl:234-235
window input flags and, ARI :569
msgWinSetLayoutDirty, ARI :249


msgWinSetPaintable, AR1:235, 237
msgWinSetTag, AR1:235
msgWinSetVisible, AR1:235
msgWinSize, AR1:246–247
msgWinSort, AR1:252
msgWinStartPage, AR1:302
msgWinUpdate, AR1:237
    image devices dirty windows and, AR1:259–260
    to tell window to repaint, AR1:239
msgWriterAppTranslator, AR1:595
msgXferGet, AR1:121, 126; AR2:31
in ASCII metrics transfers, AR2:175
in one-shot transfers, AR2:167–168, 173
    picture segments and, AR1:326–327
    in replying to one-shot transfers, AR2:176
msgXferList, AR1:121, 125; AR2:171
to list transfer types, AR2:172–173
    picture segments and, AR1:326–327
msgXferStreamConnect, AR2:178
msgXferStreamFreed, AR2:170, 177
msgXferStreamInit, AR2:178
msgXferStreamSetAuxData, AR2:170, 177
msgXferStreamWrite, AR2:169
msgXlateCanceled, AR1:606
msgXlateData, AR1:606
msgXlateMetricsSet, AR1:605
msgXlateSetFlags, AR1:605
msgZIPGetMyZone, AR2:304
msgZIPGetZoneList, AR2:304
MT (method table compiler), AWG:66; PDT:165
    object and header files and, AWG:92
    see also Method table, compiler
Multi-font component, AWG:12
Multiple access services, AR2:446
    defined, AR2:445
Multiple checklists, UI:31
    illustrated, UI:31
    scrolling, UI:187
Multiple dialog sheets, UI:205
Multiple lists, UI:184
Multiple undo model, UI:194
Multitasking service, AWG:7
Mutation capability, AR1:28

---

Name binding protocol (NBP), AR2:302–303
Name clash notes, UI:71–72
directories and, UI:72
Names
    AppleTalk protocol, AR2:302–304
    looking, AR2:303–304
    registering, AR2:303
    removing, AR2:303
    zone, AR2:304
    DOS, PDT:162
duplicate volume, AR2:50
    length of, AWG:77
    local disk volume, AR2:51
    memory-resident volume, AR2:52
    module, PDT:71
    node, AR2:53–54
    process, PDT:71
    remote volume, AR2:51
    string, PDT:82
    symbol, AWG:162–163
    task, PDT:149
    volume, PDT:27, 72
Naming conventions. see Coding conventions
NBP_CONFIRM structure, AR2:304
NBP_LOOKUP structure, AR2:303
NBP_REGISTER structure, AR2:303
Nesting, AR1:241
    of controls, AR1:363
Network disks, UI:114–115
    icon, UI:77
    Layout option sheet for, UI:115
    Options menu for, UI:114
    View menu for, UI:114
Networking, AWG:8
    facilities, AR2:250
    interfaces, AR2:253
Network printers, UI:118–119
    menus, UI:119
Network protocols, adding, AR2:251
Networks, browsing, UI:105
Network View page
    Disks section, UI:114–115
    Printers section, UI:118–119
newArgsSize, AR1:48
    _NEW_ONLY structure, AWG:50
for each class, AWG:51
    names for, AWG:51
    _NEW structure, AR2:49; AR1:17, 18
    for clsCntr, AWG:138
    contents of, AWG:118–119
    identifying, elements, AWG:51
    initializing, AWG:52
    _NEW_ONLY for each class, AWG:51
    reading, definition, AWG:50
typedef, AWG:50
New technology, UI:9–11
    new devices and, UI:10
    new operating system and, UI:10–11
new user, UI:9
    interface and, UI:11
N gesture, UI:27
    in gesture mode, UI:259
Mininote, UI:142
MiniText, UI:134
Nimbus-Q format, PDT:204–206
    AFII number array, PDT:205
    character data, PDT:205–206
    file positions, PDT:205
    converting, PDT:180
    FEDIT support of, PDT:180
    font header, PDT:205
    see also PenPoint, Packed format
Nodes, AR2:43, 52–54
    accessing, AR2:57
    attributes, AR2:54–55
    behavior of, AR2:67
    copying, AR2:80–81
    creating, AR2:43
    determining existence of, AR2:83
    flags, AR2:68
    locators and, AR2:55–56
    making, native, AR2:89–90
    moving, AR2:80–81
    names of, AR2:53
    paths, AR2:55–56
    renaming, AR2:83
    service state, AR2:396, 444, 445
    traversing, AR2:81–82
    call back routine, AR2:82
    order of traversal and, AR2:82
    quicksort routine, AR2:82
types of, AR2:52
Non-core gestures, UI:23
    application functionality and, UI:224
    for gesture mode, UI:257, 259
    listed, UI:25–27
    MiniText, UI:133–134
Nonflow pagination, AR1:137
Non-maskable interrupt (NMI), PDT:49
Non-Modal objects, UI:43
    distinction of, UI:43
Non-standard modeless command buttons, UI:179
Non-stateful objects, AWG:66
Notation conventions, PDT:86–89
    code address, PDT:88
    data address, PDT:88
    line count, PDT:88
    line numbers, PDT:88
    Scope.Identifier reference, PDT:88–89
    scope specification, PDT:86–87
    task set, PDT:89
Notebook, AWG:29; UI:14–15
    for accessing information, UI:15
    applications, UI:159
    Auxiliary, AR2:421–426
    components, AWG:29
Connections, AWG:22
Contents page, AWG:34
   for creating Empty Application, AWG:95
Contents tab, AWG:29
   for control transfer, AWG:67
default, PDT:46
directory, AWG:33
document, AWG:33
documents as one page of, UI:157
file system usage, AR2:68
Help, AWG:124–125
hierarchy, AWG:29–32
   Application Framework and, AWG:30
   application processes and, AWG:32
   file system and, AWG:31
icons, UI:76
   illustrated, UI:12
In box, UI:121
inserting documents in, AWG:104
items, AWG:33
metaphor, AWG:5; UI:14, 151
   application layer and, AWG:13
instances of, AWG:13
Notebook application, AR1:68
   activating document and, AR1:102
   Application Framework and, AR1:68
   creating document and, AR1:102
   defined, AR1:67
   document page number in, AR1:178
   file system and, AR1:89
   layout messages and, AR1:244–245
   sections, AR1:89
   table of contents, AR1:78, 89
   terminating document and, AR1:110
   NOTEBOOK APPS, AWG:34
Notebook User Interface (NUI),
   AWG:17; AR1:75, 349
   embedding documents with, AR1:77
   on-screen objects, AWG:1
   two monitors and, PDT:44
   NOTE_METRIC structure, AR1:489
   NOTE_NEW ONLY structure, AR1:16, 17
   NOTE_NEW structure, AR1:16, 17, 488
Notepaper App, AWG:265
Notepaper component, AR2:229–235
clsNotepaper view, AR2:229
data, AR2:232–234
data items, AR2:234–235
defined, AR2:229
   messages, AR2:231–232
metrics, AR2:230
Notepaper ink component, UI:255
defined, AR2:229
   notes, AR1:491
   observer, AR2:163
   string object, AR2:212
option card, AR1:516
   preference change, AR2:368
   receiving connection state, AR2:264
   scroll, AR1:453–454
   scrollbar, AR1:461–462
   string list boxes, AR1:472
   submenu, AR1:450
   toolkit table, AR1:438
   trackers, AR1:528
   Notifying, observers, AR1:52–53
   NULL-terminated strings, AR2:355
   resource agents and, AR2:353
Numbers
   frame, PDT:75
   hexadecimal, PDT:72
   line, PDT:88

OBJ_ANCESTOR_IS_A structure, AR1:55
ObjCapChk, AWG:115–116
ObjCapImp() macro, AWG:115–116;
   AR1:23–24
ObjCapOK, AWG:115–116
ObjCapRet, AWG:52, 115–116
ObjCapWarn() macro, AWG:115–116;
   AR1:23–24
objCapCall, AR1:25, 26
objCapCreate, AR1:28
   objCapCreateNotify, AR1:28, 56
   objCapFree, AR1:26
   objCapInherit, AR1:27
   objCapMutate, AR1:28
   objCapObserver, AR1:27, 29, 51
   objCapOwner, AR1:26
   objCapProp, AR1:28
   objCapScavenge, AR1:27
ObjCapSend, AR1:26, 29
ObjectCall(), AWG:45; AR1:13, 19;
   PDT:117
   address book and, AR2:320
in document reactivation, AR1:114
pData parameter and, AR1:39
updating instance data with,
AWG:140, 142, 153
OBJ_ENUM_ENUM_OBSERVERS structure,
AR1:51–52
OBJ_IS_A structure, AR1:54
OBJ_MUTATE structure, AR1:60
OBJ_NOTIFY_OBSERVERS structure,
AR1:52–53
OBJ_PROP structure, AR1:56–57
OBJ_SAVE structure, AR1:141; AR1:112
ObjWildCard, AR1:45
objWKNKey, AR1:24

Observe capability, AR1:27
Observation, AWG:27
Observed object class, AR1:70
Observer, AR1:50; PDT:14
adding an, AR1:50–51
with position, AR1:51
getting from list, AR1:51–52
messages, AR1:50, 150
text data, AR1:221
notification, AR1:50; AR2:163
byte buffer object, AR2:209
example of, AR1:53
string object, AR2:212
notifying, AR1:52–53
posting to, AR1:53
removing, AR1:51

Observing
changes, AR2:89
installation managers, AR2:407–409
tables, AR2:215, 220–221

od command, PDT:101
Odd macro, AWG:78
OK() macros, AR1:24
on access, on store commands,
PDT:101–102
on command, PDT:82–84, 129–131
access events, PDT:129
datasheet, PDT:101
fault events, PDT:130
other events, PDT:130–131
program events, PDT:129
syntax for, PDT:129
task events, PDT:129–130
variations, PDT:82

On-disk structure, AWG:174
One-shot transfer. see Transfer, one-shot
On-line documentation, AWG:13
On-screen objects, AWG:18
 Opcode, grafic, AR1:319
correct, AR1:325
opcodeMaskInvisible flag, AR1:326

Open document, UI:168–169; AR1:67
process, AR1:109
Open figures, AR1:271
drawing, AR1:293
messages, AR1:283
Open gestures, UI:236
guidelines for, UI:240
Open icon, UI:74, 221–222
Opening
address book, AR2:326
application window, AR1:196
Auxiliary notebooks, AR2:423
documents, AR1:107–109
files, AWG:143; AR2:46
for the first time, AWG:143–144
to restore, AWG:144–145
sample code, AR2:46
fonts, AR1:275–276, 303–305
parallel port, AR2:276
serial port, AR2:268
service, AR2:262–263
eXample, AR2:263
socket handle, AR2:298
see also Closing

Open service
messages sent to, AR2:470
objects, AR2:441, 470–472
clsOpenServiceObject and,
AR2:471
c1sService and, AR2:471
subclasing clsOpenServiceObject
and, AR2:471–472
see also Services

Operand specification, AWG:4
Operating system, AWG:3–18
application environment for,
AWG:20–21
Application Framework layer,
AWG:12–13
architecture, AWG:8–9
comparisons, UI:10–11
component layer, AR2:403
extensibility, AWG:14–15
kernel layer, AWG:6–7
requirements, AWG:3
Settings notebook and, UI:90
software environment elements,
AWG:41
system layer, AWG:7–12
user interface, AWG:3–5

Operator
cast, PDT:131
Delete, PDT:188
Delete hint, PDT:192
Delete Segment, PDT:186
icons, PDT:185
Merge, PDT:189
mitosis, PDT:186
Move Control Points, PDT:185
Oval, PDT:187
overloading, AWG:43
Rectangle, PDT:187
SetWidth, PDT:188
Shape Mutation, PDT:186
Shape Transformation, PDT:187–188
tilde, PDT:131
Winding Direction, PDT:188
x- and y-hint, PDT:191, 192

Option cards, AR1:511; PDT:172
current, AR1:515
Custom Resource ID, PDT:172–173
destroying, AR1:516
Exporting, PDT:172
illustrated, AR1:511
layout, AR1:515
manipulating, AR1:514–515
notification, AR1:516
painting, AR1:515–516
performance, AR1:516–517
tagging, AR1:515
TK_TABLE_ENTRY array for,
AR1:432–433

OPTION_CARD structure, AR1:169, 514
OPTION_NEW_ONLY structure, AR1:514

Options
function, UI:16
global, UI:204
In box document, UI:164
startup, UI:204

Option sheets, AWG:113; UI:44–47,
200–204; AR1:505, 511–522;
PDT:17
adding new document-wide, UI:202
adding to default document-wide,
UI:200–202
bringing up, UI:44
check gesture processing,
AR1:517–521
run-through, AR1:517–519
checklists in, UI:184
clean and dirty controls, UI:46–47
command sheets and, AR1:521–522
Controls, UI:56
creating, AR1:514
deciding when to use, UI:226
default, UI:200–202
Access sheet, UI:201
Comments sheet, UI:202
Controls sheet, UI:201
customizing, UI:201–202
design checklist and, UI:295
Disk, UI:112
displaying, AR1:517
Document, UI:89
document-wide, UI:202
dynamic behavior, UI:203–204
changing top sheet, UI:204
relationship to selection, UI:203
response to check gesture, UI:203
Empty Application, AWG:96
global application options and, UI:204
icon, UI:75
  in-line, UI:171
illustrated, AR1:511
increasing performance of,
  AR1:516-517
instead of notes, AR1:490
for keyboard accessory, UI:202
Layout, UI:88
  Connected Disks page, UI:111-112
  Connected Printers page, UI:117
  Network Printers page, UI:119
  Network View page (Disks section), UI:115
layout guidelines, UI:206-209
deactivating vs. hiding controls,
  UI:209
  non-standard layout, UI:206
  pop-up vs. in-line, UI:207-208
  standard layout, UI:206
manipulating cards and, AR1:514-517
messages, AR1:512-513
MiniNote, UI:138-139
Paper sheet, UI:138-139
Pen sheet, UI:139
MiniText, UI:43, 130-132
  multi-page, UI:43
  option tables and, AR1:521
  pages and object types, UI:46
  pop-up lists, UI:183
printer, UI:118
  for printing, AR1:138-139
  protocol, AR1:517
Quick Help and, UI:215
reference button, UI:80
resize handles for, UI:272
response to check gesture, UI:46
section, UI:89
selection in, UI:283-284
selection relationship, UI:44-45
strengths and weaknesses of, UI:225
styles of, AR1:514
summary, UI:40
table of contents, UI:87-89
usage guidelines, UI:200
user model, UI:44
uses, UI:44
vertical checklists, UI:182
  see also clsOption
Options menu, UI:44, 66-67, 192;
  PDT:172-173
Access sheet, UI:67
Comments sheet, UI:67
  for connected disk, UI:111
  for connected printers, UI:116
Controls sheet, UI:60, 66
default commands, UI:56
document-wide option sheets and,
  UI:202
FEDIT
  AutoRedraw option, PDT:191
  BezierResolution command,
  PDT:184
  View Preferences, PDT:184-185
global and startup options and, UI:204
main table of contents, UI:87
three states of, UI:87
for MiniText application, UI:45
for network disks, UI:114
for network printers, UI:119
protocol, AR1:168-169
OPTION_STYLE structure, AR1:514
Option tables, AR1:428, 521
OPTION_TAG structure, AR1:138, 167, 169
Organization
distribution volumes, AR2:390-398
file, AR2:381-398
PenPoint, AR2:382-389
required, AR2:381
Orientation, screen, PDT:40
OS_DATE_TIME structure, AR2:110
OSEnvSearch(), PDT:36
OSErrorBeep(), AR2:105
OSFastSemaRequest, AR2:414
OSEnvSearch(), PDT:36
OSFastSemaRequest, AR2:414
OSErrTime routine, AR2:110
OS.H, AR2:105
functions, AR2:105-107
OSHeapBlockAlloc(), AR1:35; AR2:95, 175
OSHeapBlockFree, AR2:88, 95
OSHeap.H, AR2:105
functions, AR2:107
OSITMsg prefix, AR2:100
OSProcessCreate, AR1:151
OSProcessHeap, AR2:472
osProcessHeapId handle, AR2:102
OSProgramInstall(), AWG:106; AR2:377, 378-379; PDT:131
OSProgramInstallate(), AWG:107; AR1:96; PDT:131
  in activating documents, AR1:102
OSSharedMemAlloc(), AR2:175
OSSubtaskCreate(), AR2:297
OSSupervisorCall() function, AR2:103
OSThisApp, AR1:399
OSTone(), AR2:105
Out, AWG:50
  arguments, AR1:14
  message header and, AWG:75
Out box, AWG:5; UI:122; AR2:305
connectivity and, AR2:244
documents in, AR2:309-310
genral device concepts, AR2:306-308
  icon, UI:13, 76
  state of application and, UI:222
  introduction, AR2:305-306
  menus, UI:122
  networking and, AWG:8
  notebook, AWG:33; AR2:308
  operation, AR2:308
  printing and, AR1:135, 302
  protocol messages, AR2:308-309
  service
  communication target, AR2:307
  enabling and disabling,
  AR2:307-308
  handling input and, AR2:312
  installing, AR2:307
  messages, AR2:313-316
  sections, AR2:306
  working with existing, AR2:311
  writing own, AR2:310
table of contents, UI:122, 160
  for customized service, UI:161
  Enabled column, UI:160
  for printer service, UI:161
  Status column, UI:160
  transfer service and, UI:159-161
  see also In box
Outer rectangle region, AR1:379
Outlined buttons, UI:176
Outline editing window, PDT:183-184
  illustrated, PDT:183
Outline fonts, AWG:9-10
  editor, AWG:14
Outline menu, PDT:183
  see thePrintManager;
  thePrintManager
Output, operation phases, AR2:311
OutRange macro, AWG:78
Overwrite boxes, UI:33
Overwrite fields, UI:33; AR1:475
  checklists with, UI:186
  editing gestures for, UI:34
  edit pads and, UI:47
  guidelines for using, UI:188
  menus with, UI:41, 199
  Proof sheet and, UI:65
  scaling, UI:269-270
  size of, UI:188
  visual segmentation cues and, UI:241
  see also Fill-in fields
Owner capability, AR1:26
Ownership
  service, AR2:257
  setting, AR2:264
Packed format. see PenPoint, Packed format

Pads, UI:47–51
- boxed/ruled, UI:50
- using, UI:51
- edit, pop-up, UI:47
- input behavior for, UI:43
- using, UI:50
- writing
  - embedded, UI:49
  - pop-up, UI:48
  - summary, UI:40

Page
- accessing, UI:15
- control, UI:157
- floating, UI:15
- menu, UI:158
- numbers, AWG:5; UI:15; AR1:509
- option sheet, UI:46
- task, PDT:152
- turning, AWG:5, 36; UI:15–16
- instead of close, AWG:37
- msgAppClose and, AWG:36
- saving state and, AWG:135

Page-level applications, AWG:33–34
- embedded applications and, AWG:35

Page-oriented documents, UI:157–158

Pages
- moving through Notebook, PDT:55

Paginating
- decisions, AR1:137
- flow documents, AR1:137
- nonflow documents, AR1:137
- paginationMethod, AR1:141

Paging control, UI:157–158
- local page numbers and, UI:158
- pop-up list on title line and, UI:158

Paint application, AWG:266–267

Painting, AWG:129; AR1:222
- border window, AR1:375–377
- background, AR1:376
- border, AR1:375
- foreground, AR1:376
- shadow, AR1:377
- buttons, AR1:422
- child windows, AR1:416
- icons, AR1:525
- labels, AR1:414–415
- list boxes, AR1:468–469
- menu buttons, AR1:447
- nesting and, AR1:241
- notes, AR1:492
- option card, AR1:515–516
- repainting as, AR1:239
- repainting as important as, AR1:236
- scrollbar, AR1:453
- stages in optimizing, AR1:241–242
- string list boxes, AR1:473

Pallette
- colors, AR1:274, 295
- compatibility, AR1:296
- construction, UI:184
- lines, UI:39
- deciding when to use, UI:226
- hiding, UI:224
- for mode switch, UI:252
- strengths and weaknesses of, UI:225

see also Accessories, palette; Pop-up palette

Paper-like visuals, UI:295

Paper option sheet (MiniNote), UI:138–139
- Line Height menu, UI:138
- Paper Width menu, UI:139

Paper Width menu (MiniNote Paper sheet), UI:139

PAPPEND utility, PDT:163–164

Paragraph attributes, AR2:9, 17–18
- changing, AR2:20
- tab, AR2:18
- changing, AR2:20

Paragraph option sheet (MiniText), UI:131

Paragraph page, UI:46

Parallel connection protocol, AR2:274

Parallel I/O interface, AR2:251–252, 275–278

Parallel port, AR2:275
- accessing, AR2:275
- cancelling printing and, AR2:278
- concepts, AR2:275
- configuration, AR2:277
- auto line feed, AR2:277
- time delays, AR2:277
- getting status and, AR2:278
- handle, AR2:276–277
- initializing printer and, AR2:277–278
- interrupts, AR2:275; PDT:50
- messages, AR2:276
- object, AR2:277
- using, AR2:276–278
- writing to, AR2:278

see also Ports; Serial port

Parent-relative sizing, AR1:398

Parent-veto layout, window, AR1:225

pArgs parameter, AWG:99, 109; AR1:20–21, 22
- defined, AR1:38
- in document reactivation, AR1:114
- in document termination, AR1:112
- msgAppMgrActivate and, AR1:103

MsgHandlerArgType() macro and, AR1:40

MsgHandler() macro and, AR1:39

MsgHandlerWithTypes() macro and, AR1:40

object and class information messages and, AR1:54

in OBJ_NOTIFY_OBSERVERS structure, AR1:52

Parity error, PDT:49

Parsing, XList data, AR1:592

Passive In box service, AR2:312

Paste command, AR1:121

Pasting, from Windows clipboard, PDT:195–196

Paths
- to Auxiliary notebooks, AR2:423–424
- in determining node existence, AR2:83
- file system, AR2:430–431
- constants, AR2:430
- function, AR2:67
- handle, AR2:85–86
- locator, AR2:55–56
- stdio and, AR2:66

Pattern descriptions, UI:243

PAUSE key, PDT:71

PCL printer, PDT:199

PDT, PenPoint on, PDT:25–62

APP.INI, PDT:44

BOOT.DLC, PDT:42–43

booting, PDT:45–49

boot sequence, PDT:29

CONSOLE.DLC, PDT:43

desktop and, PDT:55

ENVIRON.INI file, PDT:34–40

exectuting application, PDT:54–55

hardware specifications for,

PDT:25–28

initialization files, setup, PDT:28

installing application, PDT:50–53

MIL.INI, PDT:29–34

PPBOOT.EXE, PDT:29

running, PDT:25–62

stop, PDT:27

setting up specific configurations, PDT:44–45

S-Shot, PDT:178

SYSAPP.INI, PDT:44

SYSCOPY.INI, PDT:43

in tablet-like mode, PDT:40–42

universal serial pen driver, PDT:56–62

using, PDT:50

PCs
- for application edit-compile-debug cycle, AWG:14
- comparison with pen-based computers, UI:10
- kernel layer and, AWG:6

pData parameter, AWG:99, 109; AR1:35
PenPoint
Penpoint 2.0
PenPoint 1.0,
Pen-hold
Pen-down
Pen
Pen-based
Application Monitor,
compatison with disk-based systems,
direct manipulation,
updating, AWG:140
Pen, AWG:4; UI:152
comparison with mouse & keyboard,
designing for, UI:152
down point, UI:231
event, AR1:575–580
codes, AR1:568
data, AR1:575–580
low-level, AR1:559–560
gestures. see Gestures
ink mode picture, UI:248
input, UI:135
low-level, AR1:558–562
sampling, AR1:565–566
marks, AWG:9
menu (MiniNote), UI:137
modes and, UI:244–245
offsets, AR1:193
option sheet (MiniNote), UI:139
plane, AR1:296
preferences, UI:92
programming for, AWG:17
selection, UI:137
styles, UI:256
task, PDT:152
toggle switch and, UI:30–31
tracking, AWG:4
Pen-based computers, comparison with
PC’s, UI:10
Pen cursor preference, AR2:366
Pen-down gestures, UI:231
Pen-driven digitizer tablet, AWG:14
Pen-hold timeout, AWG:154
TrViewPenInput and, AWG:156
PenPoint 1.0, AWG:61
on PC, AWG:2
PenPoint 2.0
character and string constants,
AWG:62–63
character types, AWG:62
debugging, AWG:63
internationalization and localization,
AWG:61
preparing for, AWG:61–63
services, AWG:64
string routines, AWG:62
Unicode and, AWG:61
versioning data, AWG:63
PenPoint
Application Monitor, AR2:400
comparison with disk-based systems,
UI:10–11
direct manipulation, UI:151
exiting, PDT:55
facilities, AR2:250
file organization, AR2:381–398
distribution volumes and,
AR2:390–398
general structure, AR2:384
installable applications,
AR2:386–387
installable entities, AR2:386
installable services, AR2:387
internal development,
AR2:389–390
PenPoint directory, AR2:384–385
run-time services, AR2:387–388
SDK distribution, AR2:389
system distribution, AR2:385–386
Gesture font, AR2:188–190
hardware requirements, UI:5
In box and, UI:121
installed version, UI:104
interrupting, PDT:54
invoking mini-debugger on, PDT:146
not working, PDT:47
Packed format, PDT:206–210
AFII number array, PDT:207
bitmap data, PDT:210
bitmap directory, PDT:207–208
character directory, PDT:208
converting, PDT:180
file pointers, PDT:208
font header, PDT:206–207
hint data, PDT:209
long Bezier dictionary, PDT:210
shape data, PDT:209–210
short Bezier dictionary, PDT:210
see also Nimbus-Q format
running on PC, PDT:25–62
services, AR2:438–439
standard editing commands, UI:86
status sheet, UI:104
system architecture, AR2:97
volume structure, AR2:383
zooming and, UI:265
PenPoint Architectural Reference,
PDT:13–21
Application Framework section,
PDT:15
Class Manager section, PDT:13–14
Connectivity section, PDT:20
File System section, PDT:18
Input and Handwriting section,
PDT:17
Installation API section, PDT:20–21
Resources section, PDT:20
System Services section, PDT:18
Text component section, PDT:17–18
UI Toolkit section, PDT:16–17
Utility Classes section, PDT:19
Windows and Graphics section,
PDT:15–16
Writing PenPoint Services section,
PDT:21
PenPoint Class Diagram, AR1:6
PENPOINT.DIR, AR2:51
contents, AR2:54
creating, AR2:68
files, AWG:93–94; PDT:55
DOS files and, PDT:161
GDIB utility and, PDT:162
PPEND utility and, PDT:163–164
PDEL utility and, PDT:164
PSYNC utility and, PDT:164
STAMP utility and, PDT:161–162
saving memory in, AR2:77
STAMP utility and, AR2:390–391
structure, AR2:68
VPENPOINT directory, AR2:384–385
PenPoint directory, AR2:384–385
concepts, AR2:382
organization, AR2:382–383
PenPoint Installer, AR2:390
PenPointPath keyword, PDT:38
PenPoint Preferences, Zoom and Float
section, AWG:95
PENPOINT.RES, AR2:388
PenPoint UI Design Reference, PDT:13
PenProxTimeout keyword, PDT:38
Pen-tracking software, AWG:9
Pen-up event, AWG:154
[%(%]P formatting code, AWG:111
P_FS_TRAVERSE_CALL_BACK, AR2:82
P gesture, UI:27, 65
MiniNote, UI:142
MiniText, UI:134
Physical line width, AR1:340
PICSEGM data format, UI:289
PICSEG графический structure, AR1:319
PIC_SEG_HIT_LIST structure, AR1:325
PIC_SEG_METRICS structure, AR1:322
PIC_SEG_OBJECT structure, AR1:323
PIC_SEG_Paint structure, AR1:320
picSegTopGrafic, AR1:324
Pictures
appropriate use of, UI:182
list of, UI:181–182
for mode switch presentation, UI:248
with text, UI:183
see also Icons
Picture segment, AR1:279
building up, AR1:323
changing contents of, AR1:322
colors and, AR1:321
converting, format, AR1:326
copying, AR1:326–327
creating, AR1:318–319, 322
drawing in, AR1:322
drawing messages, AR1:322-323
drawing other objects in, AR1:323
editing, AR1:324
hit testing, AR1:325
moving, AR1:326-327
painting/repainting, AR1:322
storage, AR1:321
TIFF images in, AR1:333
hit testing, AR1:325
storage, AR1:321
editing, AR1:324
allocating own, AR1:258
drawing other objects in, AR1:323
drawing messages, AR1:322-323
in graphics applications, AR1:323-326
see also clsPicSeg

Pixel Paint mode, PDT:168
Back menu and, PDT:173
defined, PDT:171
Ink menu and, PDT:173
Pixels, AWG:128; UI:153; PDT:168
accessing, in image window, AR1:259-262
alignment, AR1:269
child windows and, AR1:244
clsIcon and, AR1:525
coordinate rounding errors and, AR1:269
copying, AR1:259-260
in windows, AR1:243-244
damaged, AR1:222
dirty, AR1:222
editing, PDT:194
getting and setting values, AR1:259
icon dimensions and, UI:217
icon mask and, UI:221
information, AR1:257-258
LDC, AR1:337
memory management, AR1:258
metrics, AR1:338; PDT:192
minimum number of, UI:274
nesting and, AR1:241
patterns and, AR1:291
raster operations and, AR1:292-293
repainting and, AR1:236, 238
size of, UI:274
update region and, AWG:152
updating and, AR1:241
window damage and, AR1:243
window measurement, AR1:223
Placeholder icon, UI:77
Plane mask, AR1:296
Planes, AR1:296
Plus gesture, UI:26
for contiguous selection, UI:282
in gesture mode, UI:259
guidelines for, UI:238
hot point for, UI:231
MiniNote, UI:140, 141
Pointers
data types and, AWG:76
types of, AWG:70
Pointing, AWG:4
Point path figures, AR1:342
Polygon figure, AR1:272, 339
left-hand and right-hand edges, AR1:339
Polyline, AR1:271
Pop-up accessory applications, UI:159
Pop-up choices, AR1:418, 450-451
creating, AR1:451
messages, AR1:450, 451
Pop-up edit pads, UI:47, 189
resize handles for, UI:272
Pop-up fields, AR1:475-476
Pop-up lists, UI:183
multiple modes and, UI:249
scrolling, UI:196
Pop-up menu, AR1:445
Pop-up palette
benefits, UI:252
for mode switch, UI:252
Pop-up style (embedded document), UI:170-171
closed/open illustrations, UI:170
dialog/options sheets and, UI:207-208
mixing with in-line style, UI:207-208
Pop-up writing pads, UI:48
classified pad comparison with, UI:49
resize handles for, UI:272
Portrait orientation, UI:276-277
PORTRAIT, screen device orientation, PDT:40
Ports
applications and, AR2:245
computer, AR2:245
MIL services and, AR2:246
parallel, AR2:275-278
configuration, AR2:277
interrupts, PDT:50
protocol, AR2:296
SCSI, AR2:246, 247
serial, AR2:246, 268-271; PDT:34
configuration, AR2:268-271
data modem and, AR2:279-280
writing to, PDT:136
Post-processing rules, AR1:602
defined, AR1:600
PostScript interpreter, PDT:199
printer, PDT:198
Power conservation, AWG:7
Power management preference, AR2:364
Power preferences (Settings notebook), UI:96-97
Powr task, PDT:152
PPBOOT boot program, PDT:29
during booting, PDT:46-47
p (P) commands, PDT:77
datasheet, PDT:102
P_pointer, AWG:47
PPORT_STATUS structure, AR2:278
PPORT_TIME_DELAY structure, AR2:277
P_PROC, AWG:77
Practice button (Installed Handwriting page), UI:100
prBell, AR2:365
prCharBoxHeight, AR2:366
prCharBoxWidth, AR2:366
prDateFormat, AR2:367
prDocFloating, AR2:365
prDocZooming, AR2:365
PREF_CHANGED structure, AR1:170; AR2:368
Preferences
change notification, AR2:368
time and date, AR2:110
see also System preferences
Preference settings, UI:101
Preferences Power option sheet, PDT:27
Preferences section (Settings notebook), UI:90, 91-98
Date page, UI:94
Float & Zoom page, UI:94
Fonts & Layout page, UI:93
Pen page, UI:92
Power page, UI:96-97
Sound page, UI:95
Time page, UI:95
Writing page, UI:92
pad style and, UI:48
user preference control, UI:48, 49
PrefsDateToStringO, AR2:367
PREFS directory, AR2:386
PrefsSysFontInfoO, AR2:363
PrefsTimeToStringO, AR2:367
PREF_SYSTEM_FONT_INFO structure, AR2:363
determining, AR1:536
modifying, AR1:537
region appearance, AR1:537–539
shrink-to-fit, AR1:540
style, AR1:532–533
determining, AR1:536
modifying, AR1:536
tick marks, AR1:531
value determination, AR1:538
see also clsProgressBar
PROGRESS_METRICS structure, AR1:532, 534
Progress note, UI:211; AR1:492–493
PROGRESS_PROVIDE_LABEL structure, AR1:539–540
PROGRESS_REGION structure, AR1:537
PROGRESS_STYLE structure, AR1:532
styles, AR1:533
PROGRESS_VIS_INFO structure, AR1:538
Project Scheduler, PDT:53
Proof command (Edit menu), UI:65
mark protocol and, UI:172
Proof sheet, UI:65
Properties, AR1:56–57
capability, AR1:28
creating, AR1:56–57
retrieving, AR1:57
prOrientation, AR2:363
Protected mode, AWG:93
Protection service, AWG:7
Protocol port, AR2:296
Protocols
address book, AR2:318–320
AppleTalk, AR2:301–304
blocking, AR1:168–169
embedding, UI:166–167
flow control, AR2:265
link, AR2:253
mark, UI:171–172
network, AR2:150–151
producer, AR2:168, 169
remote file access, AR2:52
RTS/CTS, AR2:252
search and replace, AR2:196
stream, AR2:168
transfer, AR2:167–170
client-defined, AR2:170
transport, AR2:253, 295
Prototype
declarations, AWG:76
function, AWG:74
Proximity, out of, PDT:38
mouse and, PDT:50
prPaginationFlow, AR1:141
prPaginationTile, AR1:141
prPenCursor, AR2:366
prPenHoldTimeout, AR2:364
prPowerManagement, AR2:364
prPrimaryInput, AR2:367
prScrollMargins, AR2:365
prSystemFont, AR2:363
prTime, AR2:366
prTimeFormat, AR2:367
prTimeSeconds, AR2:367
Pruning, AR1:441
prUnrecCharacter, AR2:368
prWritingStyle, AR2:364
Pseudoclasses, AWG:80
P_TK_TABLE_ENTRY structure, AR1:434
P_UNKNOWN, AWG:55, 77
P_WIN_METRICS pointer, AR1:233–235
q command, PDT:84
datasheet, PDT:103
QINSTALL file, AR2:386
Quadruple-Flick, UI:25
Quadruple-Tap, UI:25
in gesture mode, UI:259
MiniText, UI:133
Questionmark gesture, UI:238
Queues, intertask message, AR2:100
QUICK_DATA structure, AR2:187
PDT:19
adding gestures to, strings, AR2:191
adding, to object, AR2:181
API function, AR2:124
applications and, UI:215
clsGWin and, AR1:617; AR2:182
concepts, AR2:181–182
defined, AR2:179
defining, resources, AR2:356–357
design checklist and, UI:295
displaying, text, AR2:187
Gesture font and, AR2:188–190
gestures, AR2:182
ID, AR1:370; AR2:181
input filters and, AR1:548–549
messages, AR2:187
using, AR2:187–188
resources, AWG:167–168;
AR2:181–182
defining, AR2:183–187
defining example, AR2:184–185
definition format, AR2:183
storing ID in gesture window, AR2:186
strings, AR2:181–182
sheet, UI:13, 123
string array, AR2:183–186
window, AWG:166; AR2:181
example, AR2:186
opening, AR2:188
without clsGWin, AR2:182
see also Help
Quick Installer, UI:113; AR2:397; PDT:50
controls, UI:112, 113
Quick Install sheet, UI:113
Quicksort routine
for sorting directory entries, AR2:88–89
traverse, AR2:82
Quick start, developer’s
capturing and translating handwriting, AR1:555–558
creating a choice, AR1:352–353
creating buttons, AR1:354
creating custom layout window, AR1:353–354
creating menu bar, AR1:353
creating tabular layout window, AR1:353
file system, AR2:44–46
opening and closing files, AR2:46
reading objects and data, AR2:45
writing objects and data, AR2:44–45
handling low-level pen input, AR1:558–562
resources, AR2:337–338
simple menu with nested buttons illustrated, AR1:355
text subsystem, AR2:5
UI Toolkit, AR1:352–355
Raised buttons, UI:28, 176
RAM
capacity, UI:103
disks, PDT:27
efficiency, UI:103
file system
memory conservation and,
AWG:38–39
viewing contents of, AWG:32
item directory, AR2:376
memory-resident volume, AR2:52
for running PenPoint, PDT:25, 27, 54
volume handler, AR2:61
volumes, AWG:8
memory and, AWG:15
view, AWG:32
rand() function, AR1:25
Raster line, AR1:33
rasterOp, AWG:128
Raster operation, AR1:292–293
dynamic drawing and, AR1:293
XOR, AR1:293
RC command, AR2:359–360
returned values, AWG:47, 79–81; AR1:14
message handlers and, AWG:110
testing, AWG:80–81, 115–116
see also Status values
Revert, AWG:39
Revert command (Document menu),
Ui:57, 192
RGB color values, AR1:274, 295
compatibility, AR1:295–296
Rich Text editing, AWG:13
Right Arrow gesture, UI:25
Right-Down gesture, UI:26
guidelines, UI:240
ink editing, UI:259
MiniNote, UI:141
MiniText, UI:134
Right-Up-Flick gesture, UI:26
guidelines, UI:240
MiniText, UI:133
Right-Up-Flick gesture, UI:26
guidelines, UI:240
ink editing, UI:259
MiniNote, UI:141
MiniText, UI:133
Rings, AR2:103
Root classes, PDT:14
Root directory, AR2:52
handle, AR2:60–61
Rotate command (Edit menu), PDT:171
Routines
DB runtime, PDT:128–129
skipping execution of, PDT:123–124
routineSet specification, PDT:114
ROUTINE_SET variable, PDT:125
Rows, table, AR2:213
adding, AR2:222–223
example, AR2:223
converting number to position,
AR2:226
deleting, AR2:224
getting length of, AR2:227
getting number of, AR2:227
see also Tables
RTF, UI:289
documents, AR2:188
files, AR2:180
strings, AR2:191
RTS/CTS protocol, AR2:252
RTS (request-to-send) lines, AR2:270
Ruled pads, UI:50
compared to boxed pads and fill-in
fields, UI:50
writing, UI:48
pop-up, UI:48
ruled, UI:49
using, UI:51
Rules, AR1:600
built-in, AR1:600–601
knowledge source, AR1:601–602
post-processing, AR1:602
RULES.TXT, AWG:206, 250
Running
Adder, AWG:261
Basic Service, AWG:272
Calculator, AWG:262
Clock, AWG:264
Counter Application, AWG:196
Empty Application, AWG:94, 177
Hello World (custom window),
AWG:187
Hello World (toolkit), AWG:181
Inputapp, AWG:270
MIL Service, AWG:23
Notepaper App, AWG:265
Paint, AWG:267
PenPoint on PC, PDT:25–62
system log application, PDT:141–142
Template Application, AWG:251
Text Service, AWG:273
Tic-Tac-Toe, AWG:205
Toolkit Demo, AWG:268–269
Writerap, AWG:271
Run-Time Libraries, PDT:18
Run-time system, AR2:387–388
Sample
applications, PDT:12
profiles, PDT:115
Sample code, AWG:173–260
Counter Application, AWG:195–204
Empty Application, AWG:177–180
Hello World (custom window),
AWG:187–195
Hello World (toolkit), AWG:180–186
Template Application, AWG:251–259
Tic-Tac-Toe, AWG:204–250
Sampled images, AR1:272–274,
297–301; PDT:167
cached images, AR1:273–274,
299–301
defined, AR1:269
drawing, AR1:273
messages, AR1:297
operator, AR1:297–299
call backs, AR1:299
drawing, AR1:298
filter, AR1:298
image model, AR1:299
rendering colors, AR1:299
run-length encoding, AR1:298
Saving
Adobe font, PDT:199–200
characters and bitmaps subset,
PDT:196
counter object, AWG:145–146
data, AWG:135-148
document, AWG:153
instance data, AR1:35
state, AWG:37, 135
typing, PDT:80-81
Scalability
    design checklist and, UI:295
designing for, UI:153-154
Scope Spec, PDT:86-87

Scribble, editing window component, AR1:555
translating and displaying, AR1:557-558

Scavenging
    object, AR1:607
    defined, AR1:607
    messages, AR1:608
    attribute, AR1:608, 609
    notification, AR1:608, 610
    stroke, AR1:608, 609-610
    objects, AR1:597
    Pen menu and, UI:137
    rendering, AR1:607
    translator notification, AR1:608
    using, AR1:607-610
see also clsScribble

Scripts, AWG:162
    actions of, AR1:453-462; PDT:16
    bubble, AR1:453
    drag handle, AR1:453
    frames and, AR1:453
    layout, AR1:453
    notification, AR1:454
    offset, AR1:454
    range, AR1:456
    painting, AR1:456
    providing information to, AR1:454
    scroll windows, AR1:457-462
    thumbing, AR1:454
    updating, AR1:456

SCROLLBAR_SCROLL structure, AR1:454-455
    offset, AR1:455-456
    Scroll function, UI:16

Scrolling, UI:16, 263-265
to beginning/end, UI:36, 264
by flicking, UI:264-265
by screenful, UI:36, 263
by unit, UI:36, 263
guidelines for, UI:263-265
horizontal, UI:274-275
line to top/bottom, UI:264
list boxes, AR1:466
lists, UI:186-188
checklists, UI:160
gesture accelerators for, UI:188
multiple checklists, UI:187
pop-up checklists, UI:196
of selectable items, UI:187
marked location, UI:172
normalizing, AR1:456-457
with Tap & Flick, UI:37
text view, AR2:29
with thumbing, UI:36
thumbing vs., AR1:455-456
vertical, AR1:455
window, inserting text view in, AR2:30

Scroll margins, UI:35-37, 56

Arrow & Drag Box style, UI:263-264

components, UI:36
control (Fonts & Layout preferences), UI:93
gestural, UI:37
hiding, UI:56
horizontal, UI:35
illustrated, UI:55
operations, UI:35, 263-264
showing and hiding, UI:264
style preference, AR2:365
Tap & Flick style, UI:37, 264
vertical, UI:35

Scrollwin, AR1:457
adding windows to, AR1:462
alignment, AR1:459
creating, AR1:458-459
inner window, AR1:459, 460
layout, AR1:460-461
multiple windows in, AR1:462
notification, AR1:461-462
repainting, AR1:460
style flags, AR1:459
windows, AR1:459
wsSynchRepaint flag and, AR1:242
see also Scroll windows

SCROLL_WIN_DELTA structure, AR1:461

Scroll windows, AR1:451-462
creating scrollwin, AR1:458-459
layout, AR1:460-461
multiple, AR1:462
notification, AR1:461-462
repainting, AR1:460
scrollwin windows, AR1:459
toolkit tables, AR1:462
see also clsScrollWin; Scrollwin

SCROLL_WIN_STYLE structure, AR1:461

SDK, AWG:1, 14
    contents, PDT:22-23
    compiler tools, PDT:23
    optional goodies, PDT:23
    PenPoint, PDT:22
    directory, AR2:385, 389
    contents, AR2:389

documentation
    API Reference, PDT:21
    Application Writing Guide,
PDT:11-12
    Architectural Reference, PDT:13-21
    Development Tools, PDT:13
    documents in, PDT:10
    feedback on, PDT:10
    library, PDT:9
    organization of, PDT:13
    sample applications in, PDT:12
    suggested approach to, PDT:9-10
    UI Design Reference, PDT:13
    see also Documentation

files, AWG:67
organization of, AWG:1
outline font editor, AWG:14
source-code symbolic debugger, AWG:14

Search and replace, AR2:195–198
API function, AR2:124; PDT:19
concepts, AR2:195
highlighting text and, AR2:198
messages, AR2:198
protocol, AR2:196
replacing characters and, AR2:198
searching text and, AR2:197
setting initial search position, AR2:196
writing class and, AR2:196–198

Search and replace, AR2:195–198

Search (Find & Replace sheet), UI:63

Searching
address book, AR2:326–328
more information, AR2:327–328
search query, AR2:326–327
search result, AR2:327
searching text and, AR2:197

Selecting,
Select function, Select All command (Edit menu), UI:62
deselecting and, UI:230
deleting, PDT:186
shape mutation and, PDT:186
transitions, AR2:157
user model and, UI:278
Selection manager, AWG:11–12; AR2:155–163; PDT:19
client messages, AR2:158–159
determining selection, AR2:157
function, AWG:154; AR2:124
messages passes to, AR2:161–162
messages sent to selection owners, AR2:159–161
observer notification and, AR2:163
preserving, AR2:156
preserving selection and, AR2:156
protocol, AR2:157–158
selection transitions and, AR2:157
self Parameter, AR2:138
object UID and, AR1:35
Self (UID), AWG:56–57
EmptyAppDestroy and, AWG:110
message handler parameter, AWG:99, 109
SEL_OWNERS structure, AR2:161–162, 163
Sendable services, AR2:331–334

Send (Find & Replace sheet), UI:63

Sending
messages, AWG:45–48; AR1:13–15
to window hierarchy, AR1:251–252

Send user interface, AWG:13

Self
defaults, AR2:269
requesting and releasing, AR2:268

Serial driver, AR2:265
Serial handle
checking, AR2:271
defaults, AR2:269
requesting and releasing, AR2:268

Serial I/O interface, AR2:251, 265–274
buffered data, AR2:265
concepts, AR2:265–266
concurrency issues, AR2:266
events, AR2:266
flow control, AR2:265–266
high-speed packet I/O concepts, AR2:273–274
interrupt driven I/O, AR2:265
messages, using, AR2:267–273
detecting events, AR2:272–273
flow control, AR2:272
reading/writing with serial port, AR2:271
reinitializing serial port, AR2:268
requesting/releasing serial handle, AR2:268
sending BREAK, AR2:272
serial port configuration, AR2:268–271
Service applications, UI:159
SERVICES directory, AR2:384, 386, 395
contents, AR2:387
files in, AR2:396
INST directory and, AR2:396
SERVICE.INI file, AR2:256, 385, 387
service directory and, AR2:444
Service installation manager, AR2:416
class, AR2:441
Service instances, AR2:439–440
creating, AR2:442, 454–455
dynamic, AR2:453–454
file system node, AR2:443
msgSvcSaveRequested and, AR2:466
preconfigured, AR2:473
saving state data and, AR2:466
static, AR2:453–454
using, AR2:442–443
Service manager, AR2:250, 251, 256–257
accessing service and, AR2:261
architecture, AR2:437
binding to service and, AR2:262
class, AR2:440
defined, AR2:256
finding handle and, AR2:263–264
finding service and, AR2:261–262
function, AR2:255, 256, 437
messages, AR2:260
change ownership protocol, AR2:467–469
from, AR2:461–466
handled by clsService, AR2:469–470
notification, AR2:259, 260
sent by, AR2:459–470
opening and closing service and, AR2:262–263
predefined, AR2:258–259
receiving state notification and, AR2:264
request for information, AR2:459–461
services and, AR2:440
setting service owner and, AR2:264
unbinding from service and, AR2:263
using, AR2:261–264
Service messages, AR2:443
Service.RES file, AR2:387
service directory and, AR2:444, 445
Services, UI:163–165; AR1:76–77;
AR2:438, 438–439
accessing, AR2:258–259
overview, AR2:257
service manager and, AR2:261
binding to, AR2:259
broken connection and, AR2:247
chaining, AR2:446
classes, AR2:255–264, 439–441
initializing, AR2:451, 452
closing, AR2:263
clsStream messages and, AR2:470
connections, AR2:258, 446–447
connection status, AR2:224, 447
data storage, AR2:455
defined, AR2:250, 255
deinstalling, AR2:256, 456
design decisions, AR2:449
directory contents, AR2:444
distributing, AR2:473–474
documentation and, AR2:474
providing demo application, AR2:473
providing preconfigured instances, AR2:473
DLL and DLC files and, AR2:256
exclusive access, AR2:445–446
file system and, AR2:443–445
in file system at run time, AR2:445
In box and Out box, AR2:305
devices and, AR2:306–307
enabling and disabling, AR2:307–308
installing devices and, AR2:307
passive and active, AR2:312–313
sections, AR2:306
installable, AR2:387
installation, AR2:378–379, 450–456
calling initialization routines, AR2:451–452
calling InitService, AR2:452–453
calling other class initialization routines, AR2:452
creating service instances, AR2:454–455
static and dynamic service instances, AR2:453–454
tasks and, AR2:455–456
installation to use overview, AR2:441–443
installing, AR2:256
interfaces and, AR2:249–250
layered, AR2:248
messages sent to open, AR2:470
MIL, AR2:246–249, 439
see also Parallel port; Serial port
msgSvcGetMetrics response, AR2:460
msgSvcSetMetrics response, AR2:461
multiple access, AR2:445, 446
multiple volumes and, AR2:398
multi-user, AR2:446
msgSvcOpenDefaultsRequested and, AR2:466
non-port, AR2:439
opening, AR2:258–260
on distribution/boot disks, AR2:444
owner, AR2:247
setting, AR2:264
ownership, AR2:225
task, AR2:455–456
preconfigured, AR2:396
printer, AR2:250
programming, AR2:449–472
service manager and, AR2:440
shared, AR2:446
targeting, AR2:258, 446
test examples, AR2:476–485
BASICSVC, AR2:485–487
MILSVC, AR2:487–480
TESTSVC, AR2:476–485
in theSelectedVolume, AR2:445
transaction, AR2:296
requesting, AR2:300
responding to, AR2:301
unbinding, AR2:263
upgrading, AR2:398
writing, AR2:435–472
see also Open service

Services Architecture, AWG:64; PDT:21
Service sections, AR2:236
Service State Nodes, AR2:396, 444, 445
setbufO system service, AR2:65
Set string message, AWG:42–43
Settings
icon, UI:13, 76
powering down and, UI:97

Settings notebook, UI:90–106
configuration modification and, UI:164
contents, UI:91

Fonts and Layout page
Field Font, UI:271
Hand Preference, UI:35
PenPoint Font, UI:271
scroll margin selection, UI:263
Tap & Flick, UI:37
Top Edge control, UI:276

Gesture Timeout, PDT:38
Installed Software section, UI:90, 97–101
Applications page, UI:98
Installed dictionaries page, UI:100
Installed Fonts page, UI:101
Installed Handwriting page, UI:99–100
Installed Services page, UI:164
Installed User Profiles page, UI:101
Installer, PDT:41
organization, UI:90
overview, UI:90–91

Preferences section, UI:90, 91–98
Date page, UI:94
Float & Zoom page, UI:94
Fonts & Layout page, UI:93
Pen page, UI:92
Power page, UI:96–97
Sound page, UI:95
Time page, UI:95
Writing page, UI:92
scaling and resizing and, UI:268
Status section, UI:90, 102–104
PenPoint status sheet, UI:104
Storage Details sheet, UI:103
Storage Summary sheet, UI:102
table of contents, UI:91
Install button, UI:99
Writing Timeout preference, UI:33
see also Connections notebook; Help notebook; Notebook;
Stationary notebook

Windows, UI:90, 91–98
 MeiID preferences, UI:90–95

ỗi 404: Page not found.
Settings notebook. see Auxiliary notebooks
SetUpFont() function, AR1:592
SetWidth, PDT:180
change, PDT:188
character placement and, PDT:181
S gesture, UI:27, 64
MiniNote, UI:142
MiniText, UI:134
Shadow Gap style field, AR1:377
SHADOW_NEW_ONLY structure, AR1:509
Shadows, AR1:509
creating, AR1:509
painting, AR1:377
turning on, AR1:378
Shapes. see Character shapes
Sheets
dialog, UI:40
input behavior for, UI:43
modal, UI:178
modelless, UI:178
multi-page, UI:43
option, UI:44–47
summary, UI:40
summary, UI:40
see also Layout option sheet; Option sheets
Shift command (Edit menu), PDT:171
Show menu (System Log), PDT:142
Show Special Characters control (MiniText), UI:132
ShowText() function, AR1:592
Shrink-wrap, AR1:397
constraints and, AR1:395–396
parent-relative sizing and, AR1:398
points to watch for with, AR1:395
window layout and, AR1:382
Shutdown button (Power preferences), UI:96, 97
Signature pads, scaling, UI:269–270
Signed data types, AWG:76
Simple class, AR1:82–84
instance of, AR1:84
Single-stepping, PDT:77
SIO.Control_IN_STATUS structure, AR2:270
SIO.CONTROL_OUT_SET structure, AR2:270
SIO EVENT_HAPPENED structure, AR2:272–273
SIO EVENT_SET structure, AR2:272
getting current, AR2:273
SIO_FLOW_CONTROL_CHAR_SET structure, AR2:270
SIO_FLOW_TYPE structure, AR2:269
SIO_INIT structure, AR2:268
SIO_INPUT_BUFFER_STATUS structure, AR2:271
SIO_LINE_CONTROL_SET structure, AR2:269
SIO METRICS structure, AR2:270–271
SIO_OUTPUT_BUFFER_STATUS structure, AR2:271
Size menu, PDT:173
SIZEOF type, AWG:77
SM.ACCESS structure, AR2:261
Small icons, UI:74
SmartDrive, PDT:27
smartAttrStationaryMenu, AR1:153
SM_BIND structure, AR2:262
in unbinding from service, AR2:263
SM CLOSE structure, AR2:263
SM.CONNECTED_NOTIFY structure, AR2:264
SM.FIND_HANDLE structure, AR2:263–264
SM.OPEN_CLOSE structure, AR2:262–263
accessing socket and, AR2:298
SM_SET_OWNER structure, AR2:264
Snapshots, PDT:175
file name, PDT:176
full-screen, PDT:177
of gestures, PDT:177
impossible, PDT:177
taking, PDT:177
before, PDT:176
writing to disk, PDT:177
Socket, AR2:295–296
accessing, AR2:298
closing, handle, AR2:299
connection, AR2:296
identifier, AR2:296
transport address, AR2:296
SoftTalk interface, AR2:250, 295
Software
development environment, AWG:15–16
hierarchy, AR1:76
Sort By menu (table of contents), UI:88
Sorting, directory entries, AR2:88–89
Sound preferences (Settings notebook), UI:95
Sound routine, AR2:105
Source application, AWG:11
Source code
additional, AWG:90
debugging, PDT:72–73
Empty Application, AWG:99–101
setting breakpoint and, PDT:74
system debugger, AWG:14
viewing, PDT:77–78
Source code file, AWG:76
beginning of, AWG:99
organization, AWG:97–99
application C code file, AWG:98–99
message handler parameters, AWG:99
method table file, AWG:98
structure, AWG:73
Source debugger. see Source level debugger (DB)
Source level debugger (DB), AWG:133;
PDT:67
activation of, PDT:84
advanced techniques, PDT:123–131
cast operator, PDT:131
compile time commands, PDT:124
core inside breakpoints, PDT:131
controlling execution threads,
PDT:124
install and start commands,
PDT:131
on command, PDT:129–131
skipping execution, PDT:123–124
tilde operator, PDT:131
wait command, PDT:131
built-ins, PDT:125–129
predefined types, PDT:125–126
runtime routines, PDT:128–129
useful values, PDT:126
useful variables, PDT:127–128
command line editing, PDT:81
command reference, PDT:85–112
command datasheets, PDT:89–112
command summary, PDT:85–86
notation conventions, PDT:86–89
compiling and linking and,
PDT:69–70
certainty, PDT:73
executing code at prompt, PDT:79
executing, PDT:84
features, PDT:67
files used in session, PDT:69
Hello World (custom window) and,
AWG:134
installing, PDT:70
invoking, PDT:71
memory use and, PDT:81–82
messages, objects, status values,
AWG:162
mini-debugger and, PDT:145
profiling, PDT:113–121
clearing profile information,
PDT:121
code profiling, PDT:113–117
displaying profile information,
PDT:119–120
object profiling, PDT:117–118
profile breakpoints, PDT:113
specific tasks, PDT:121
type of profiles, PDT:113
for saving typing, PDT:80–81
scripts, PDT:81
to send messages, PDT:82
speeding up debugging with,
AWG:133–134
for suspected code problems, AWG:87
tag name, AWG:70
using, AWG:2; PDT:71–84
  breakpoints, PDT:73–74
txt command, PDT:73
examining and setting values,
PDT:75–77
executing C code, PDT:79–80
g command, PDT:71
hexadecimal numbers, PDT:72
memory use and, PDT:81–82
for message sending, PDT:82
module names, process names, task
IDs, PDT:71
PAUSE key, PDT:71
prompting circumstances, PDT:84
saving typing, PDT:80–81
single-stepping, PDT:77
source code debugging, PDT:72–73
st command, PDT:74–75
string names for messages, objects,
statuses, PDT:82
tl command, PDT:80
u command, PDT:78–79
v command, PDT:77–78
watching memory, PDT:82–84
see also Debugging; Mini-debugger
Spatial modes, UI:244
Speaker, modem, AR2:284–285
Spell checking, UI:64
Spell command (Edit menu), UI:62, 64
mark protocol and, UI:172
Spelling dictionary, AR1:601
Spell sheet, UI:64
modifying and replacing, UI:194
Spreadsheet, embedded chart in, UI:167
Square buttons, UI:28, 176
scred command, PDT:72–73, 77
datasheet, PDT:103
srcRect, AR1:244
SR_GET_CHARS structure, AR2:197
SR_REPLACE_CHARS structure, AR2:198
S-Shot utility, AWG:174; PDT:175–178
bugs, PDT:178
installing, PDT:175
on Macintosh, PDT:177
on PC, PDT:178
specifying a file name, PDT:176
taking snapshot, PDT:177
using, PDT:175–177
hists on, PDT:177–178
specifying a delay, PDT:176
specifying an area, PDT:175–176
window, PDT:175
Stack trace, PDT:149
Stak task, PDT:152
STAMP command, AWG:93–94
for EMPTYAPP, AWG:94
Stamping, AWG:164
applications, AWG:93–94
Stamp mapping, AR1:131–132
STAMP utility, AR2:390–391;
PDT:161–162
eample, PDT:162
stationary and, AR2:392
syntax for, PDT:161–162
Standard application menus (SAMS),
AWG:33–34; AR1:163–170
chck gesture handling, AR1:169–170
clsCntrApp, AWG:146
document and edit menus,
AR1:165–168
frames and, AR1:504–505
message handling and, AWG:108
options menu protocol, AR1:168–169
see also Menu
Standard message interface, AR1:492–496
defined, AR1:487
kinds of, AR1:492–493
Standard modal command buttons,
UI:178–179
Standard modeless command buttons,
UI:178
StartApp keyword, PDT:39
start command, PDT:131
Starting point, locator, AR2:55
Stateful objects, AWG:89
creating, AWG:65
separate, AWG:150
States, AWG:135
application class, AR1:96
diagram of, AR1:95
browser, AR2:143
connection, notification, AR2:264
created, AR1:100, 101
document, AR1:100
diagram of, AR1:100
transitions, AR1:101
dormant, AR1:100, 101
filing, AWG:135
rule, AWG:136
list boxes, AR1:467
saving, AWG:135
Tic-Tac-Toe, AWG:150
STATIC keyword, PDT:115
Stationary, AWG:5, 163–165; UI:120;
PDT:54
application, AR2:392–393
specific, AWG:38
building, AWG:165
custom, UI:120
design checklist and, UI:295
directory, UI:162
documents, UI:162; AR2:377
removing document to, AR2:427
icon, UI:13, 76
installable, UI:162
loading and unloading, AR1:153–154
menu
adding document to, AR2:426
modifying, AR2:426–427
palette, AWG:104
pop-up menu, AWG:23, 163
illustrated, AWG:164
templates, AR2:387
Tic-Tac-Toe source files, AWG:250
Stationary notebook, AWG:22–23, 33;
UI:13, 120; AR2:393
applications in, AWG:104
document installation and, UI:162
document stationary, AR1:152
Empty Application documents and,
AWG:95
illustrated, AWG:164
table of contents, UI:120
see also other notebook types
STATNRY directory, AWG:164, 205;
AR2:387, 392–393
Tic-Tac-Toe stationary and, AWG:165
STATUS, AWG:77, 82
codes, input event, AR1:564, 569
gauges, UI:13
index, AWG:169
Status-checking macros, AWG:87
Status section (Settings notebook), UI:90,
102–104
PenPoint status sheet, UI:104
Storage Details sheet, UI:103
Storage Summary sheet, UI:102
Status values, AWG:47, 79; AR1:11–12
in accessing service, AR2:261
checker, AWG:82
coding convention for, AWG:72
defined, AR1:11
defining, AWG:80
generic, AWG:81
human-readable, AWG:81
less than stsOK, AWG:52
message handlers and, AWG:110
pseudoclasses for, AWG:80
STATUS, AWG:77, 82
testing returned, AWG:80–81
see also Returned values
st command, PDT:74–75
datasheet, PDT:104–105
in mini-debugger, PDT:148
StdErrorO, AWG:168–169; AR1:494
buttons and, AWG:170
resource files and, AWG:170
stdio calls, AR2:65
STDIO.h, AR2:113
sibling, AR2:99
see also Tasks
SuperScriptII, sample definition, PDT:61
Suspend
button (Power preferences), UI:96
capability, UI:96–97
SVC_BIND structure, AR2:462
SVC_GET_SET_METRICS structure
msgSvcGetMetrics and, AR2:459–460
msgSvcSetMetrics and, AR2:460
SVC_INIT_SERVICE structure, AR2:452–453
SVC_NEW_ONLY structure, AR2:454–455
SVC_OPEN_CLOSE structure, AR2:463
SVC_OWNEd_NOTIFY structure, AR2:468
SVC_TERMINATE_VETOED structure, AR2:458
Swap
boot, PDT:39
file, PDT:39
error messages and, PDT:47
SwapBoat keyword, PDT:39
SwapFileSize keyword, PDT:39
Switch
mode, UI:247
location, UI:251–252
presentation, UI:248–250
statements, AWG:76
two-state, UI:198
see also Toggle, switches
Symbolic debugger, AWG:85
Symbols
generating automatically, AWG:162
names of, AWG:162–163
printing, AWG:163
sym command, PDT:72, 82
datasheet, PDT:105
Synchronous message passing, PDT:14
SYSAPP.INI, AR2:385; PDT:44
in boot sequence, PDT:29
description, PDT:28
SYS_BOOT_STATE structure, AR2:431–432
SYSCOPY.INI, AR2:385; PDT:44
in boot sequence, PDT:29
description, PDT:28
files, PDT:43
SYSDC_CACHE_IMAGE structure, AR1:300
SYS_DC_COPY_IMAGE structure, AR1:300
sysDcDrawDynamic, AWG:128
sysDcDrawDynamic flag, AR1:293
sysDcDrawFast flag, AR1:293
sysDcDrawRectangle, AR1:212
SYSDC_FONT_ATTR structure, AR1:305
SysDcFontID function, AR1:303
SYSDC_FONT_METRICS structure,
AR1:307–308
SysDcFontString function, AR1:303
sysDcHoldDetail, AR1:291
sysDcHoldLine, AR1:291
SYSDC_IMAGE_INFO structure,
AR1:297–298
SYSDC_LINE structure, AR1:291
SYSDC_MODE structure, AR1:292–293
sysDcPatNil, AR1:294
SYSDC_PIXEL structure, AR1:292–293
sysDcRopCopy, AR1:292–293
sysDcWindingFill flag, AR1:293
SYSDC_FONT_METRICS structure,
AR1:307–308
Log Size, PDT:143
SysFont, PDT:143
Show, PDT:142
Trace, PDT:142–143
on PC, PDT:142
running, PDT:141–143
using, AWG:112
System-modal note, AR1:487–488
application-modal note vs., AR1:490
System notebooks, AWG:5
System preferences, AR1:170;
AR2:361–368
auto shutdown, AR2:365
auto suspend, AR2:364
date format, AR2:367
date and time, AR2:364
disabled, AR2:367
floating allowed, AR2:365
gesture timeout, AR2:364
hand preference, AR2:363
handwriting times, AR2:364
line height, AR2:366
list of, AR2:362
observer of, AR2:368
pen cursor, AR2:366
power management, AR2:364
press-hold timeout, AR2:364
primary input device, AR2:367
resources, AR2:361
screen orientation, AR2:363
scroll margin style, AR2:365
scrolling allowed, AR2:365
time, AR2:366
unrecognized character, AR2:368
writing style, AR2:364
 zooming allowed, AR2:365
Systick rate, setting, PDT:128
Syst task, PDT:152
Tab
bars, AR1:428
adding items to, AR1:508
creating, AR1:508
layout, AR1:508
windows, AR1:223
clicking and, AR1:230
setting, AR1:183–184
see also Tabs
TAB_BAR,...NEW_ONLY structure, AR1:308
Table class, AR2:213–228
concepts, AR2:213–215
distributed DLL, AR2:213
messages, AR2:217–218
Table data
files, AR2:214
getting, AR2:223–224
setting, AR2:223
Table layout, AR1:383
constrains, AR1:386–387
flags, AR1:384–385
sample, AR1:383
specifying, AR1:385–386
structure, AR1:384–385
using tAlignBaseline for, AR1:388
see also clsTableLayout
Table-like mode, PDT:40–42
Table mapping, AR1:131
Table object
access concurrency characteristics, AR2:216
accessing, AR2:216
creating, AR2:220
current state, AR2:215
observing, AR2:215
Table of Contents, AWG:5
bookmark check box, AR2:145
browser and, AR2:137–138
Connections notebook, UI:105
creating, AR2:138
default object, UI:46
document icons in, UI:216–217
edit pads and, UI:47
embedding documents and, UI:20
Empty Application title, AR2:18
Help notebook, UI:124
icons, UI:74
hot points and, UI:217
In box, UI:121
main, UI:87
menu line, UI:85
multiple documents and, UI:86
Notebook, UI:12, 15
checkboxes in, UI:185
menus, UI:85–87
option menus, UI:87–89
zero or one list style, UI:185
option sheets, UI:87–89
Document, UI:89
icon, UI:75
Layout, UI:88
Out box, UI:122, 160
for customized service, UI:161
Enabled column, UI:160
for printer service, UI:161
Status column, UI:160
Settings notebook, UI:91
Install button, UI:99
Stationary notebook, UI:120
Table of Contents (TOC) application, AWG:13
Tables, AR2:213
accessing, AR2:214
beginning, AR2:221
ending, AR2:228
adding rows to, AR2:222–223
Boolean operators for, AR2:225
creating, AR2:221
defined, AR2:213
defining, AR2:218–219
descriring, AR2:213–214
discontiguous selection in, UI:283
dragging in, UI:288
files, AR2:214
freeing, AR2:228
horizontal scrolling and, UI:274–275
locating records in, AR2:215
messages for, AR2:217–218
information, AR2:226
observing, AR2:215, 220–221
positioning in, AR2:215
Quick Help and, UI:215
searching, AR2:224–226
selection feedback and, UI:279
semaphores and, AR2:222
shared, AR2:215–217
access to table object, AR2:216
concurrency, AR2:216–217
ownership, AR2:216
state, AR2:227
using, in database, AR2:217
see also Columns, table; Rows, table
TableServer, AWG:15
Tabs, UI:15
area, AWG:34
Show Special Characters control (MiniText), UI:132
toggling, UI:32
see also Tab
Tab Stop option sheet (MiniText), UI:132
Tabular layout window, AR1:353
TA_CHAR_ATTRS structure, AR2:16–17
TA_CHAR_MASK structure, AR2:16–17
tagAppMgrDefaultDocName, AR1:98
Tag argument, AR2:16
atomChar, AR2:16
atomPara, AR2:17
atomParaTabs, AR2:18
TagNum() macro, AR1:12
tagPrAutoShutdown, AR2:365
tagPrAutoSuspend, AR2:364
Tags, AWG:79; AR1:12
for data transfer types, AR2:166–167
jumping to structure definitions with, AWG:118
for string array resource, AWG:167
window, AR1:216, 225
TAGS subdirectory, AR2:389
TA_MANY_TABS structure, AR2:18
Tap & Flick scroll margins, UI:37, 264
TA_PARA_MASK structure, AR2:17
Tap gesture, UI:16, 24
Double, Triple, Quadruple, UI:25
in gesture mode, UI:259
guidelines for, UI:239
for zooming, UI:266
family, UI:24
summary, UI:235
in gesture mode, UI:258
guidelines for, UI:236
hot point for, UI:231
for ink mode selection, UI:255–256
MiniNote, UI:140, 141
MiniText, UI:133
for selection, UI:279
Tap & Flick scroll margins and,
UI:37, 264
see also Gestures
Tapping, UI:17
boxed pads and, UI:50
clean and dirty controls and, UI:46–47
for deselection, UI:280
for drag & drop cancellation, UI:69
edit pads and, UI:189
Help notebook, UI:124
icons, UI:74
Installed Dictionaries page, UI:100
Installed User Profile page, UI:101
in menu fields, UI:199
menus and, UI:42
MiniNote and, UI:135
multi-page sheets and, UI:43
multiple checklists, UI:31
on fill-in field, UI:33
on page number, UI:15
pop-up checklists and, UI:30
Power preferences and, UI:96–97
Proof sheet and, UI:65
Quick Help sheet, UI:123
reference buttons and, UI:20, 79
for scrolling, UI:36
for selection, UI:279
Spell sheet and, UI:64
subpage controls and, UI:38
tab, UI:15
toggle switch and, UI:10–181, 30
zero or one style and, UI:185
for zooming, UI:267
Tap-Press gesture, UI:16, 24
in gesture mode, UI:258
guidelines for, UI:237
for initiating drag, UI:286
for ink editing, UI:255–256, 258
MiniNote, UI:135, 140, 141
for zooming, UI:266
Target, AR1:129
directory, AR2:59
changing, AR2:86
input, AR1:572–573
objects, AR1:548, 550
defined, AR1:550
record, AR1:130
service, AR2:258

Targeting, UI:231–233
auto-selection and, UI:283
communication devices, AR2:307
drag destination, UI:288
guidelines for, UI:232–233
selection, UI:232
services, AR2:446
unselected objects, UI:232
window objects, UI:233
zone, UI:232

Task
commands, PDT:85
events, PDT:129–130
IDs, PDT:71, 121
name, PDT:149
process, 0; PDT:153–154
profiling, PDT:121
set, PDT:89
stack position, PDT:86
terminating, PDT:124
values available for, PDT:126
see also specific tasks; Tasks

Task list, PDT:80
mini-debugger, PDT:151–154
every example, PDT:149–150
system tasks in, PDT:151–152

Task List (Show menu), PDT:142

Task management, AR2:98–99
priority level, AR2:99
processes, AR2:98
software task scheduler, AR2:99
subtasks, AR2:98–99

Tasks, AR2:98
80386 protected mode and, AR2:102
family, AR2:99
priority level, AR2:99
privilege level, AR2:103
services and, AR2:455–456
software, scheduler, AR2:99
see also Subtasks; Task

TASK_SET variable, PDT:126

TBL_BEGIN_ACCESS structure, AR2:220, 221
TBL_BOOL_OP, AR2:225
TBL_COL_DESC structure, AR2:218–219
TBL_COL_GET_SET_DATA structure
going data and, AR2:223
setting data and, AR2:223
TBL_CONVERT_ROW_NUM structure,
 AR2:226
TBL_CREATE structure, AR2:218
TBL_FIND_ROW structure, AR2:224–225
TBL_FREE_BEHAVE values, AR2:220
TBL_GET_SET_ROW structure
going data and, AR2:224
setting data and, AR2:223
TBL_GET_STATE structure, AR2:227–228
TBL_LAYOUT_CONSTRAINT structure,
 AR1:386
TBL_LAYOUT_COUNT structure, AR1:385
TBL_LAYOUT_INDEX structure, AR1:388–389
TBL_LAYOUT_SIZE structure, AR1:386
table layout constraints and, AR1:386
TBL_LAYOUT_STYLE structure, AR1:384
TBL_NEW structure, AR2:220
TBL_ROW_POS value, AR2:215
TBL_STRING structure
going data and, AR2:224
setting data and, AR2:223
TBL_TYPES, AR2:219
TD_METRICS structure, AR2:14
TD_NEW structure, AR2:13
Template Application, AWG:251
classes, AWG:251
compiling, AWG:251
files, AWG:252
running, AWG:251–252
sample code, AWG:251–260
tutorial, AWG:90
TEMPLATE.RC, AWG:252, 259
Templates
constraining with, UI:242–243
levels of, UI:243
Template services, AR2:449–450
TEMPLTAP.C, AWG:252, 253–256
TEMPLTAP.H, AWG:252–253
Terminating
applications, AWG:38–40
documents, AWG:36–37;
 AR1:110–113

Testing
application, PDT:44
Basic Service, AWG:272
Hello World, AWG:272
Hello World (custom window),
 AWG:187
Hello World (toolkit), AWG:181
MIL Service, AWG:273
return values, AWG:115–116
Test Service, AWG:273
Test Service, AWG:272–273
TESTSVC service, AR2:476–485
defined, AR2:475
METHOD.TBL, AR2:476
OPENOBJ.H, AR2:483–485
TESTSVC.C, AR2:477–483
TESTSVC.H, AR2:477

Text
boxes, UI:32, 33
character encoding, AR2:8
characters, AWG:9
characters in, AR1:312
component, PDT:17–18
collection with graphic
subsystem, PDT:18
draws, UI:233
MiniText and, UI:130
scaling, UI:269–270
targeting unselected objects and,
 UI:232
targeting zone, UI:232
default object, UI:46
dragging in, UI:288
drawing, AR1:311–312
fields, UI:32–34, 188; PDT:16
fill-in, UI:33
fill-in vs. overwrite, UI:188
font, UI:188
gestures in, UI:34
overwrite, UI:33
text boxes, UI:32, 33
fitting, AR1:313
formatting, UI:16
index, AR2:27–29
insertion pads, AR2:9
creating, AR2:33
destroying, AR2:33
messages, AR2:33
using, AR2:33
length, AR2:14
list, UI:181–182
measuring, AR1:312
metrics, AR2:14
for mode switch presentation, UI:248
moving word in, UI:69, 291
option sheet, UI:16
with pictures, UI:183
selection, UI:283
feedback for, UI:279
move marquee for, UI:287
spacing, AR1:312
splicing, AR1:313
style, getting and setting, AR2:32
substituting, AWG:170
subsystem, AWG:155
views, AWG:40
widths, AR1:309
see also Text subsystem; Text views

Text attribute arguments, AR2:16–20
character attributes, AR2:16–17
paragraph attributes, AR2:17–18
paragraph tabs, AR2:18

TEXT_BUFFER structure, AR2:14, 15
TEXT_CHANGE_ATTRS structure,
 AR2:19–20
changing defaults and, AR:1:434–435
child windows, AR:1:434
class-dependent creation information, AR:1:429–433
common creation information, AR:1:428
displaying installed fonts, AR:1:441
flags, AR:1:430–432
modifying items with, AR:1:435
values, AR:1:431
installed fonts and, AR:1:441
kinds of, AR:1:427–428
layout, AR:1:437
list boxes vs., AR:1:470
managers, AR:1:438–440
messages, AR:1:427
modifying, AR:1:436
notification, AR:1:438
painting, AR:1:436–437
removing items from, AR:1:441
sample, AR:1:426
scroll windows and, AR:1:462
toggle tables, AR:1:427, 442
window tags, AR:1:436
see also clsTKTable; UI Toolkit

Tools
accessory palette, AWG:22
debug, PDT:67
development, PDT:5–7

Top Edge menu (Fonts & Layout Preferences), UI:93

Topping, border windows, AR:1:377
TOPS SoftTalk (Sitka), AR:2:253, 295
TOPS software, AWG:8
TP_NEW_ONLY structure, AR:2:298
TP_RECVFROM structure, AR:2:300
TP_SENDCVTO structure, AR:2:300
TP_SEN DTO structure, AR:2:299

Trace menu (System Log), PDT:142–143:

Tracing, AWG:159–160

Trackers, AR:1:527–528; PDT:17
destroying, AR:1:528
drawing, AR:1:528
notification, AR:1:528
see also clsTrack

Transaction, AR:2:199
aborting, AR:2:204–205
adding items to, AR:2:203–204
beginning, AR:2:202–203
data, AR:2:201
ending, AR:2:204
history, changing size of, AR:2:205
metrics, getting, AR:2:205
services, AR:2:296
requesting, AR:2:300
responding to, AR:2:301
undoing, AR:2:206

Transfer
ASCII metrics, AR:2:175
buffer
fixed-length, AR:2:174
structures, AR:2:167
types, AR:2:173–174
variable-length, AR:2:174–175
client-defined, AR:2:170
defined, AR:2:167
format supports, AWG:11–12
functions and messages, AR:2:170–171
one-shot, AR:2:167–168
defined, AR:2:167
performing, AR:2:173–174
replying to, AR:2:176
operations, AWG:8
protocols, AR:2:167–170
stream, AR:2:168–170
defined, AR:2:167
performing, AR:2:176–178
types, AR:2:166
adding to list, AR:2:172–173
establishing, AR:2:171–173
listing, AR:2:172
requesting, AR:2:172
searching list, AR:2:173
Transfer class, AR:2:165–178; PDT:19
concepts, AR:2:165–167
establishing transfer type,
AR:2:171–173
functions, AR:2:124
messages and, AR:2:170–171
performing one-shot transfers and,
AR:2:173–176
performing stream transfers and,
AR:2:176–178
transfer protocols, AR:2:167–170
Transfer services, UI:159–161
connected, UI:160
defined, UI:159
disconnected, UI:160
table of contents, UI:160–161
modifying, UI:161

Translate & Edit command (MiniNote Edit menu), UI:137

Translating, captured scribbles, AR:1:557–558

Translation
boxed pads and, UI:50
data structures, AR:1:598–600
XLATE_CASE_METRICS, AR:1:599
XLATE_METRICS, AR:1:598–599
XLATE_NEW, AR:1:599
XLATE_NEW_ONLY, AR:1:599
defined, UI:253
using ink and, UI:255
handwriting, flags, AR:1:600–602
immediate, UI:253
improving, UI:242
input and handwriting, AWG:10–11
messages, AR:1:604–606
control messages, AR:1:606
creating translator, AR:1:605
initialization messages, AR:1:605
notification messages, AR:1:606
modes, UI:250
ruled pads and, UI:50
score, AR:1:600
templates, AR:1:602–603
of text strings, AWG:63
Translation classes, AR:1:597–606
data structures, AR:1:598–600
handwriting translation flags,
AR:1:600–602
hierarchy of, AR:1:598
translation messages, AR:1:604–606
translation templates, AR:1:602–603
Translation template, AR:1:601
modes, AR:1:603
types, AR:1:602–603
Translator, AR:1:479
classPaper, AR:1:595–596
creating, AR:1:605
notification, AR:1:608
object, AR:1:555, 597
setting, AR:1:587–588
see also Handwriting, translators

Transparency (color) values, AR:1:295

Transport
address, AR:2:296
binding to, AR:2:301
protocols, AR:2:253
service types, AR:2:296
Transport API, AR:2:295–304
concepts, AR:2:295–297
using clsTransport, AR:2:297–301
for AppleTalk, AR:2:301–304

Traversal drivers, AR:1:129

Traverse
back routine, AR:2:82
ordering of, AR:2:82
quicksort routine, AR:2:82

Triple-Flick, UI:25

Triple-Tap, UI:25

in gesture mode, UI:259
guidelines for using, UI:239
MiniNote, UI:141
MiniText, UI:133
zooming and, UI:266

tsAlignEdge, AR:2:29

TSR (terminate and stay resident), PDT:152

TSS task, PDT:152
tt command, PDT:124
t (T) commands, PDT:77
datasheet, PDT:106

TTTAPP.C, AWG:206, 210
command line editing, PDT: 81
DB.INI file, PDT: 80–81
using DB scripts, PDT: 81
TZ keyword, PDT: 39

u command, PDT: 78–79
datasheet, PDT: 108–109
U gesture, UI: 27
MiniNote, UI: 142
MiniText, UI: 134

UI components, AR1: 349
UIDs (unique identifiers), AWG: 49
administered value, AWG: 53
class, AWG: 102–103
component, AR1: 203
creating new objects and, AR1: 15
dynamic, AR1: 9
filing, AWG: 143
going and setting, AR1: 181
going viewed object and, AR2: 26
identifying bits, AWG: 102
input subsystem, AR1: 568
for new object identification,
AWG: 52–53
open service object class and, AR2: 441
for published applications, AWG: 175
of root container application, AR1: 187
saving, AWG: 143
scope and type, AR1: 10
testing, AWG: 103

Undo
command, AR2: 199
Edit menu, UI: 62, 194
gesture, UI: 16, 24
in gesture mode, UI: 258
guidelines for, UI: 237
hot point for, UI: 232
history, AR2: 201
items, AR2: 199, 201
deallocating buffer and, AR2: 201

UNDO_ITEM structure, AR2: 203
Undo manager, AR2: 199–206; PDT: 19
concepts, AR2: 199–201
deallocating buffer and, AR2: 201
function, AR2: 124
instance, AR2: 200
messages, AR2: 202
using, AR2: 202–206

UNDO_METRIC structure, AR2: 205
Undo redo model, UI: 194

Unfilled region (progress bar), AR1: 531
defaults, AR1: 537
manipulating, AR1: 537–539
Unicode, AWG: 61
UniPen, PDT:56
custom syntax, PDT:57–59
notes on using, PDT:60–62
sample definitions, PDT:61–62
UNIPENPORT tag (MILINI), PDT:60
UNIPENPROTOCOL tag (MILINI), PDT:61
UNIPENTYPE tag (MILINI), PDT:60
predefined types, PDT:60
Unique identifier (UID), PDT:14
dynamic, PDT:14
numbers, PDT:14
well known, PDT:14
UNISTD.H, AR2:114
Units, for scrolling, UI:259
Up-Right gesture, UI:26
Up-Left gesture,
Up-Down gesture,
Update region, AR1:222
Unknown errors, AR1:495
Universal serial pen driver,
UniPen, PDT:56
Unrecognized character preference,
AR2:368
Unsigned data types, AWG:76
Update region, AR1:222, 240
msgWinBeginPaint and, AR1:240
msgWinBeginRepaint and, AR1:240
smart repainting and, AR1:238
updating ends and, AR1:240–241
Up-Down gesture, UI:259
Up-Left gesture, UI:26
guidelines for, UI:240
in MiniNote, UI:142
Up-Right gesture, UI:26
guidelines for, UI:240
in MiniNote, UI:142
in MiniText, UI:133
U...routines, AWG:62
User column, browser, AR2:145–146
User dictionaries, UI:100
User interface, AWG:3, 18
access paths and, UI:171
application layer and, AWG:13
automatic layout and, UI:153
bitmap editor, PDT:170–173
built with windows, AR1:218
buttons in, UI:151, 171
class and object use in, AWG:20
comparisons, UI:11
consistency in, UI:154
design, UI:149–150
consistency and, UI:149
development process and, UI:5
direct manipulation and, UI:151
graphic, UI:149
iterative development and, UI:151
wording and, UI:151
design guidelines, AWG:17–18
designing, AWG:60
development from, AWG:18
displaying, AR2:332
fields and, AR1:485
frames and, AR1:507
layering, UI:224
layout speedup and, AR1:365
model, UI:245–246
modes and, UI:244–245
notebook metaphor, AWG:5; UI:151
overview, UI:12–20
pen, AWG:4
resizing elements of, AWG:10
resource files and, AR1:365–366
see also UI Toolkit
User Interface Toolkit, AWG:10
User models, UI:245–246
drag & drop, UI:285–286
for input modes, UI:246
for moving and copying, UI:285–294
selection and, UI:278
see also Drag & drop interface
User profiles, UI:101
icon, UI:78
installed, UI:101
installable software views and, UI:109
Users, comparison of, UI:9
Utilities
GDIR, PDT:162
GO, PDT:165
MAKELABEL, PDT:162–163
MT, PDT:165
PAPPEND, PDT:163–164
PDEL, PDT:164
PSYNC, PDT:164
RC, PDT:165
RESAPPND, PDT:165
RESDUMP, PDT:165
S-Shot, PDT:175–178
STAMP, PDT:161–162
Utility classes, AR2:123; PDT:19
features, AR2:124
see also specific classes
UUIDs (universal unique identifiers), AR1:122
component, AR1:203
embedded window, AR1:193
getting and setting, AR1:181
mark, AR1:201
uv command, PDT:109–110
V86a task, PDT:152
V86x task, PDT:152
Values
attribute, AR2:78
getting length of, AR2:79
setting, AR2:79
types of, AR2:76
zero, AR2:77
choice, AR1:443–444
current, AR1:403
examining and setting, PDT:75–77
? command, PDT:75–76
identifier types, PDT:76
known identifiers, PDT:76
lexical scope, PDT:77
getting, AWG:139–140
incrementing, AWG:140
useful, in DB, PDT:126
xtmMode, AR1:603
xtmType, AR1:602–603
varArgs functions, AWG:169
Variables, AWG:70
DB useful, PDT:127–128
debugging flag sets, PDT:134–135
setting, PDT:127
vars command, PDT:111
v command, PDT:77–78
datasheet, PDT:110
Verbs use, in button labels, UI:175
ver command, PDT:111
Versioning data, AWG:63
Version keyword, PDT:39
Version number, object, AR1:56
Versions, DLL file, AR2:402
operating system, AR2:403
Vertical Flip command (Edit menu), PDT:171
VGA video adapter, PDT:25
View
data object and, AWG:155
dumping, AWG:161
Tic-Tac-Toe, AWG:150
data interaction, AWG:152–153, 155
msgGWInGesture, AWG:156
selection and, AWG:153–155
tracking and, AWG:154
window coordinates, AWG:152
View class, AWG:26, 27–28; AR1:69,
173–175
concepts, AR1:173
Tic-Tac-Toe, AWG:150
see also Classes; clsView
View menu, UI:196
MiniText, UI:130
samples of, UI:196
table of contents, UI:86
network disks, UI:114
network printers, UI:119
VIEW_NEW structure, AR1:174
View Preference dialog (FEDIT Options menu), PDT:184–185
illustrated, PDT:185
Views, for displaying data, AR1:78
“Virtual keyboard,” AWG:11
Visual segmentation cues, UI:241
spatial modality and, UI:244
Vmajor (minor), AWG:126
VolSel keyword, PDT:40
VOLSEL line, AWG:97
Volume connectivity strategy, AR2:244
Volumes, AR2:43, 49–52
  boot, PDT:29
  concepts, AR2:49
  connecting and disconnecting, AR2:50
  defined, AR2:49
  directory structure on, AR2:53
  distribution, AR2:390–398
  general structure, AR2:384
  getting information about, AR2:90
  labelled, PDT:27–28
  list of, AR2:49
  messages specific to, AR2:91
  multiple, AR2:51
  names, AR2:51
  of, PDT:27
  DOS, PDT:72
  duplicate, AR2:50
  local disk, AR2:51
  memory-resident, AR2:52
  remote, AR2:51
  setting/changing, AR2:91
  protection of, AR2:67
  selected, PDT:37
  selection, PDT:54
  traversing, AR2:82
  types, AR2:50–52
  local disk, AR2:51
  memory-resident, AR2:52
  remote, AR2:51–52
  uses of, AR2:381
Volume structure, AR2:383
vu command, PDT:111–112

WACOM510, sample definition, PDT:62
WACOM510C, sample definition, PDT:62
Wacom pen tablet, PDT:45
wait command, PDT:131
Warning() function, AR1:23

WATCOM, AWG:66
C/386 compiler, AWG:92
C/386 compiler and linker, PDT:69–70
C/386 runtime functions, AWG:7
  compiler and linker flags, AWG:92–93
Make files, PDT:148
OS/2 linker, AWG:67
  protected mode applications and, AWG:93

WATCOM C run-time library, see C
run-time library
Well-defined gestures, UI:236
  guidelines for, UI:238
Well-known list resource IDs, AR2:344
  defined, AR2:343
  index, AR2:344
Well-known ports, AR2:296
Well-known resource IDs, AR2:343
  defined, AR2:343
Well-known UIDs, AR1:9–10
  administration of, AR1:10–11
  creating, AR1:10
  for development and testing, AR1:11
  global, AR1:9, 10–11
  OBJECT_NEW_ structure, AR1:47
  private, AR1:10, 11
  process-global, AR1:10
  scope of, AR1:9, 10
White space, in graphic design, UI:149
Wild cards, AWG:98
  method table, AR1:45, 46
WIN_COPY_RECT structure, AR1:243
WIN_DEV_PIXELMAP structure, AR1:258
Winding direction, PDT:188
  merging and, PDT:189–190
  viewing and altering, PDT:188
Window, AR1:209
  address, AR2:332
  creating, AR2:333
  filling, AR2:333–334
  application, AR1:196–197
  baseline, AR1:387–388
  alignment, AR1:251, 387
  bitmap edit, PDT:194
  borders, AR1:361
  bounds, AR1:232
  setting, AR1:234
  character selection, PDT:182
  child, AR1:216
  altering, AR1:246
  labels and, AR1:415–416
  client, AWG:27; AR1:92
  closing document and, AR1:109
  creating for application frame, AR1:212–213
  document termination and, AR1:112
  frame layout and, AR1:501
  positioning scroll window, AR1:461
  clipping, AR1:216, 219–221
  regions, AR1:220
  copying pixels in, AR1:243–244
  creating, AR1:212, 217
  size and position, AR1:232
  style flags, AR1:232
  custom, AWG:27
  damaged, AR1:222
  copied pixels and, AR1:243

debugging, AR1:252
  decoration, AR1:382
  defined, AR1:215
  delta, AR1:377
  destination, AR1:259–260
  destroying, AR1:218
  devices, AR1:210, 217
  cached images for, AR1:273–274
  setting, AR1:233
  dirty, AR1:222, 237
  image devices and, AR1:259
  marking entire, AR1:239
  receiving msgWinRepaint and, AR1:237
  region, AR1:238–239
  drawing context bound to, AR1:210
  drawing in, AWG:133; AR1:236
  embedded, AR1:117–118
  child, AR1:193
  creating, AR1:190
  destroying, AR1:190
  metrics, AR1:190
  moving or copying, AR1:191–193
  between, AR1:118–119
  style of, AR1:190–191
  toolkit ancestors and, AR1:370
  UUID, AR1:193
  enumerating, AR1:216, 225
  environment information, AR1:253
  extracting, AR1:218, 234
  filling, AR1:216, 225, 253
  filling, AR1:294
  flags, AR1:227, 228–229
  input, AR1:228, 569–570
  setting, AR1:234–235
  style, AR1:228, 229, 232
  floating, AWG:27; AR1:163
  font attribute, PDT:197
  font header, PDT:196
  gesture, AR1:368–370
  messages, AR1:617–618
  using, AR1:617–618
  graphic classes and, AR1:210–211
  graphics subsystem and, AR1:546
  grouping, AR1:223
  hint editing, PDT:190–191
  illustrated, PDT:191
  image, AR1:259–262
  ImagePoint, AR1:210
  creating and inserting
  AR1:556–557
  initialization, AWG:131–132
  initializing, AR1:560–561
  inserting, AR1:217, 233–234
  layout, AWG:122; AR1:216, 224–225, 247–248, 381–382
  adding child windows to, AR1:381
  classes, AR1:381–382
  dirty, AR1:249, 365
  episodic, AR1:225
  parent-veto, AR1:225
  processing, AR1:249
illustrated, AR1:217
inserting window into, AR1:550
objects, AR1:548
update region, AWG:152
see also Window system

Window class, AWG:27; UI:233;
AR1:227–234
classHelloWin, AWG:125
classWin messages, AR1:230–231
classWin structures, AR1:227
creating new window and, AR1:232
enhancements, AWG:121–122
Hello World (custom window) and, AWG:89
layout messages, AR1:244–251
management messages, AR1:251–253
for storing numeric value, AWG:136
summary, AR1:254
window display messages,
AR1:235–244
window metrics messages,
AR1:233–235
for window organization, AWG:122
see also Classes; clsWin

Window device classes, AR1:255–263
image devices, AR1:256–263
windowing devices, AR1:255
see also clsWinDev

Windowing, AWG:8–9
device, AR1:255
defined, AR1:255

Windows clipboard
copying to, PDT:195
pasting from, PDT:195–196

Window system, AR1:215–226
advanced repainting strategy and,
AWG:152
caching desired sizes and, AR1:250
concept overview, AR1:215–226
embedded applications and, AWG:35
enumeration options, AR1:225
imaging devices and, AR1:256
layout and geometry capture, AR1:251
layout episode, AR1:249
repaint algorithm, AWG:155
in repaint process, AR1:242–243
smart repainting and, AR1:238–239
subclassing clsWin and, AR1:226
windowing devices and, AR1:255
see also Window

win.input.flags field, AR1:565
WIN_METRICS structure, AR1:227–228,
254
caching desired sizes and, AR1:250
child windows and, AR1:246
laying out self and, AR1:250
window metric messages and,
AR1:233–235
WinMode keyword, PDT:40

WIN_NEW structure, AR1:232
WIN_SEND structure, AR1:251
WIN_SORT structure, AR1:252
Wipe-through, UI:280
WkNAdmin() macro, AR1:12
WkItemResId() macro, AR2:344
WkListResId() macro, AR2:344
WkObjResId() macro, AR2:345
WkResId() macro, AR2:345
WKNScope() macro, AR1:12
WKValue() macro, AR1:12
WkNVer() macro, AR1:12
WLINK command file, AWG:93

Wording
of button labels, UI:210–211
fundamentals, UI:151
guidelines, UI:210–211
of messages, UI:210

Word lists, UI:242

Word processor
embedded icon in, text, UI:168
large embedded icon in, text, UI:168
open embedded document in, text,
UI:169

Work area, UI:56
icons, UI:74
illustrated, UI:55

Work space, UI:102

Wrapper document, AR2:310
Wrapper. see Printing, wrapper

Writerap, AWG:271–272
WriterAppInit0, AR1:595
WriterAppTranslator0 method, AR1:595
WriterCompleted0, AR1:592

Writing
address book, AR2:328–330
agents, own, AR2:354
browser state, AR2:143
data resource, AR2:349–350
resource agents and, AR2:353
to debugger stream, PDT:137–139
files, AR2:83
to log file, PDT:135–136
object resource, AR2:351
objects and data, AR2:44–45
to parallel port, AR2:278
preferences (Settings notebook), UI:92
to second monitor, PDT:136–137
with serial port, AR2:271
to serial port, PDT:136
with service, AR2:435–472
streams, AR2:134
with timeout, AR2:134–135
style preference, AR2:364
see also Reading
Writing pads
boxed/ruled, UI:50
embedded, UI:49
pop-up, UI:48
Writing Paper application, AR2:31
Writing Timeout, UI:93
preference, UI:33
wsCaptureGeometry flag, AR1:247, 251
wsChildrenStay flag, AR1:244
wsClipChildren flag, AR1:236
wsClipParent flag, AR1:236
wsClipSiblings flag, AR1:236
wsClipParent flag, AR1:236
wsClipSiblings flag, AR1:236
wsClipParent flag, AR1:236
wsClipSiblings flag, AR1:236
Comment input, AR1:326
XferStreamAccept() function, AR2:177
XferStreamConnect() function, AR2:176
function, AR2:176–177
xferString, AR1:126
xg1Tap gesture, AR1:408
xg2Question gesture, AR1:408
xlate argument, AR1:479
XLA TE_CASE_METRICS structure, AR1:599
XLA TE.H, AR1:598
XLA TE_METRICS structure, AR1:598–599
XLA TENuevo structure, AR1:599
XLA TE Nuevo structure, AR1:599
xList2Text filter function, AR1:592
xList, AR1:597
concepts, AR1:611–612
creating, AR1:614
data, AR1:588
functions, AR1:588
parsing, AR1:592
defined, AR1:611
elements, AR1:611–612
adding, AR1:615
data, AR1:612
deleting, AR1:614
flags, AR1:612
freeing, AR1:614
getting and setting, AR1:615
inserting, AR1:614
flags, AR1:611
functions, AR1:613
using, AR1:613–615
msgXlateData and, AR1:606
traversing, AR1:614–615
XListDelete() function, AR1:614
XLIST_ELEMENT structure, AR1:611–612
XListFreeData() function, AR1:614
XListFree() function, AR1:614
XListGet() function, AR1:615
XList.H, AR1:611
XListInsert() function, AR1:614, 615
XListNew() function, AR1:614
XListSet() function, AR1:615
XListTraverse() function, AR1:592,
614–615
XON/XOFF flow control, AR2:265–266
see also Flow control
Your comments on our software documentation are important to us. Is this manual useful to you? Does it meet your needs? If not, how can we make it better? Is there something we’re doing right and you want to see more of?

Make a copy of this form and let us know how you feel. You can also send us marked up pages. Along with your comments, please specify the name of the book and the page numbers of any specific comments.

Please indicate your previous programming experience:

- [ ] MS-DOS
- [ ] Mainframe
- [ ] Minicomputer
- [ ] Macintosh
- [ ] None
- [ ] Other __________________

Please rate your answers to the following questions on a scale of 1 to 5:

- How useful was this book?
- 1 Poor 2 Average 3 Excellent

- Was information easy to find?
- 1 Poor 2 Average 3 Excellent

- Was the organization clear?
- 1 Poor 2 Average 3 Excellent

- Was the book technically accurate?
- 1 Poor 2 Average 3 Excellent

- Were topics covered in enough detail?
- 1 Poor 2 Average 3 Excellent

Additional comments:

________________________________________________________________________

________________________________________________________________________

________________________________________________________________________

________________________________________________________________________

________________________________________________________________________

________________________________________________________________________

Your name and address:

Name ________________________________________________________________

Company ______________________________________________________________

Address ________________________________________________________________

City _________________________ State ________________ Zip ________________

Mail this form to:

Team Manager, Developer Documentation
GO Corporation
919 E. Hillsdale Blvd., Suite 400
Foster City, CA 94404-2128

Or fax it to: (415) 345-9833
PenPoint™ Development Tools

PenPoint Development Tools describes the PenPoint development environment and the tools that you use to develop, test, and debug PenPoint applications. The first part introduces you to PenPoint application development, describes the organization of the PenPoint SDK documentation set, and instructs you how to run PenPoint on a 80386/486 PC. The second part describes how to debug PenPoint programs; this includes a tutorial on the PenPoint source-level debugger. The third part covers the tools provided with the PenPoint SDK, including file system tools, a bitmap editor, and a font editor.

The PenPoint operating system is a compact, 32-bit, fully object-oriented multitasking operating system designed expressly for mobile, pen computers. GO Corporation designed the PenPoint operating system as a productivity tool for people who may never have used computers before, including salespeople, service technicians, managers and executives, field engineers, insurance agents and adjustors, and government inspectors.

Other volumes in the GO Technical Library are:

PenPoint Application Writing Guide provides a tutorial on writing PenPoint applications, including many coding samples.
PenPoint User Interface Design Reference describes the elements of the PenPoint NoteBook User Interface, sets standards for using those elements, and describes how PenPoint uses the elements.
PenPoint Architectural Reference, Volume I presents the concepts of the fundamental PenPoint classes.
PenPoint Architectural Reference, Volume II presents the concepts of the supplemental PenPoint classes.
PenPoint API Reference, Volume I provides a complete reference to the fundamental PenPoint classes, messages, and data structures.
PenPoint API Reference, Volume II provides a complete reference to the supplemental PenPoint classes, messages, and data structures.

GO Corporation was founded in September 1987 and is a leader in pen computing technology for mobile professionals. The company's mission is to expand the accessibility and utility of computers by establishing its pen-based operating system as a standard.

GO Corporation
919 East Hillsdale Blvd.
Suite 400
Foster City, CA 94404

Addison-Wesley Publishing Company