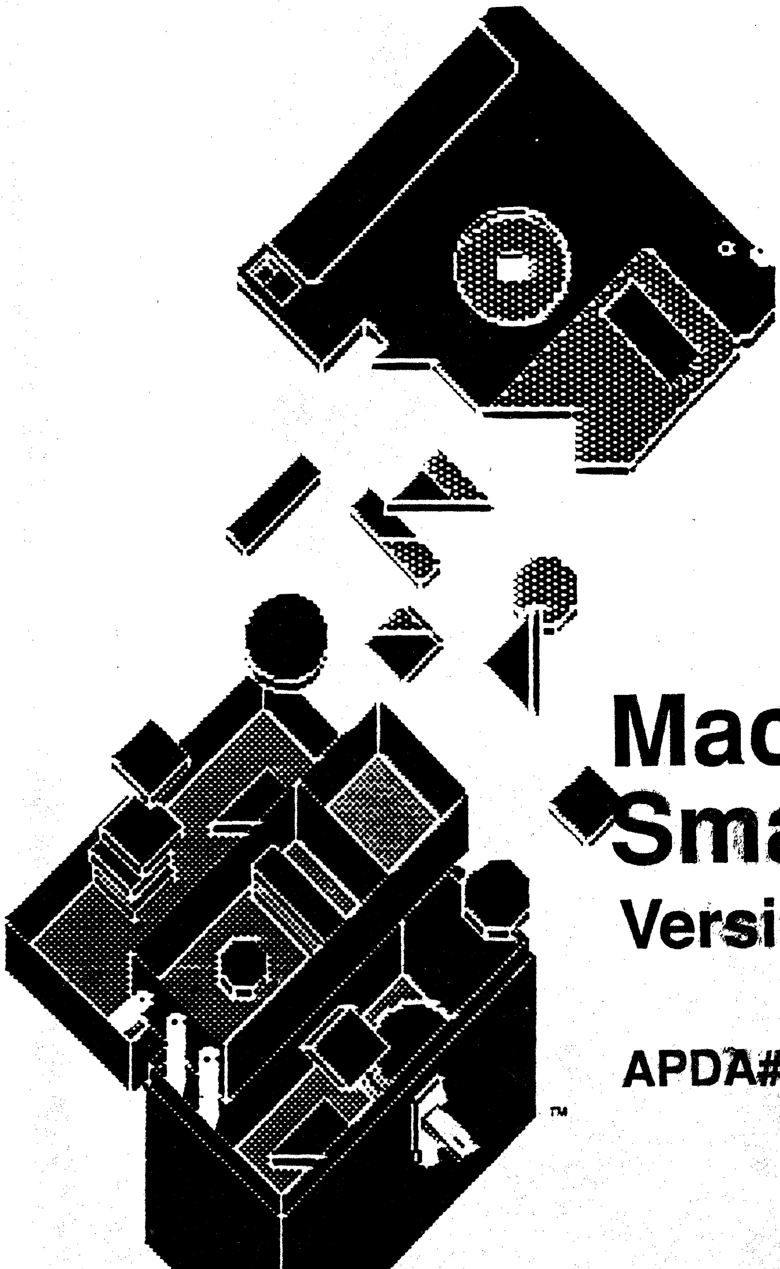
APPLE
PROGRAMMER'S
AND DEVELOPER'S
ASSOCIATION



Macintosh SmallTalk-80™

Version 0.4

APDA# KMSST4

Smalltalk-80™ for the Macintosh™ 0.4 Release Note

June 9, 1987

Overview:

The main purpose for the 0.4 release is to provide a Smalltalk that runs well on the Macintosh SE and Macintosh II (as well as the Macintosh Plus and other Macintosh models with at least one Megabyte of RAM). Since Smalltalk is implemented using some non-standard techniques, a few changes had to be made to Smalltalk 0.3 to make it work on the new machines. While we were making these changes we fixed a few other bugs and added several features. The Smalltalk 0.3 documentation still applies except as noted in this release note.

Warning: Images and the interpreter from this release are not compatible with those from version 0.3 or earlier versions. Any work you want to move between 0.3 or earlier Smalltalk versions and this 0.4 Smalltalk release must be moved via file-ins.

Note: *You must run with version 4.1 or greater of the System file.* System 4.1 is provided with the release. This does not apply to the Macintosh XL; however, on the Macintosh XL you should be able to run with System 3.2 (which you were probably using already).

Installing the release:

In general, follow the directions in the Smalltalk 0.3 manual except as noted below.

If you do not have an up-to-date system (System file 4.1 or greater) on your hard disk, boot from Smalltalk Disk 1 and move the System file and Finder from Smalltalk Disk 1 into your System Folder. If you wish to preserve the fonts and desk accessories in your old System file you should use the Installer to install System 4.1.

If you already have the 0.3 Smalltalk-80.sources you don't need new ones (they have not changed for 0.4). If you don't already have the sources, use DivJoin to join the Smalltalk sources on Smalltalk Disk 3 and Smalltalk Disk 4. The Smalltalk sources must be placed either in your root directory or in the directory where you will run Smalltalk. If you plan on running Smalltalk from more than one directory, place the sources in the root.

To install this release on a Macintosh XL, you will have to move the files to single-sided disks before the Macintosh XL can read them. To do this, you will have to join the larger files using a Macintosh with 800K drives then divide them again onto single sided disks.

Differences for Installation on a hard disk: (refer to pages 3-4 of the 0.3 manual)

1. Smalltalk-80.sources is now a two part file on Disks 3 & 4. It used to be a four part file. You will still have to join the file.
2. You no longer have to join Smalltalk.image. It is now a complete file on Disk 2.

Differences for Installation without a hard disk: (refer to pages 4-6 of the 0.3 manual)

1. Most of the process described in the 0.3 release notes tells you how to create two double sided disks named **System** and **Work**. The configuration in which Smalltalk 0.4 is shipped on double sided disks makes Disk 1 already set up as the **System** disk and Disk 2 already set up as the **Work** disk. All you need to do to run the 0.4 release without a hard disk is make a copy of Disk 1 and call it **System** then make a copy of Disk 2 and call it **Work**. You can now follow step 7 (on page 6 of the 0.3 manual) to boot Smalltalk 0.4 from the **System** and **Work** disks. If you want to make more room for your Smalltalk.changes file on your **System** disk, you can remove **Divjoin 1.0d9** and the **New Goodies** folder. You can also remove the LaserWriter and ImageWriter files from your System Folder . If you really want more space, you can then remove some of the fonts and desk accessories from the system file using the Font/DA Mover. Make sure you do all this removing on a backup copy in case you later need something you had removed.

Packing List:
This 0.4 release includes four 800K release disks. They contain the following files:

Smalltalk Disk 1 (1 of 4)
- System Folder
- Smalltalk.changes
- DivJoin 1.0d9
- New Goodies (folder)

Smalltalk Disk 2 (2 of 4)
- Smalltalk.interp
- Smalltalk.image

Smalltalk Disk 3 (3 of 4)
- 1.Smalltalk-80.sources

Smalltalk Disk 4 (4 of 4)
- 2.Smalltalk-80.sources
- Goodies-Demos (folder)
- Goodies-Utilities (folder)

Summary of Improvements:

Bugs Fixed:

1. The implementation of the Smalltalk class DisplayScreen and its support within the image have been totally redesigned and reimplemented. Now Smalltalk can again use the top scanline on the screen and still run on a Macintosh II.
2. The way Smalltalk handles calls to the VIA has been totally redesigned and reimplemented.
3. The system will no longer get confused when you open more than one transcript in a project. There is no reason to have more than one transcript per project, so you are no longer allowed to do that.
4. The bundle bit is now set in the interpreter file, so when you double click on an image, the interpreter will be started.
5. Bugs were fixed in the MacPaint.st, Macintosh-Quickdraw.st and rs232.st goodies.
6. A confusing cancel message that came up when users tried to throw away a window with unaccepted changes in it has been reworded to make more sense.
7. Some file system bugs were fixed.

New Features:

1. Files created from Smalltalk will still be type "TEXT", but their application signature is now "MPS " like files created by MPW (previous releases of Smalltalk used MacWrite's signature). This way, if you double-click on one of these files you will enter the MPW shell (instead of MacWrite). If you don't have MPW you can edit these files by opening them from within any word processor or text editor using the File menu. The application signature for the file Smalltalk-80.sources was not changed for this release since the file itself has not changed since the 0.3 release.
2. The implementation of pop-up menus has been changed to make them faster.
3. From a file list, you can now set the MacDefault directory by using the **set default** item in the top menu. From the second menu in a file list, you can now open a changes browser by selecting the **browse changes** item. This is very useful when you are trying to integrate several people's changes, since the changes browser contains an option to select conflicts. This will show you the methods you are about to file in that are already included in your changes list. These are the ones that have already been changed by a file-in contributed by someone else.
4. If you hold the Shift key down when you select a context in the debugger, it will decompile the method rather than reading the source. This is handy if you're having file system problems.

5. There are three new goodies. They are in the "New Goodies" folder on Disk 1.
   a. The QDViews.st goodie provides users with a simple and relatively robust way of performing QuickDraw operations within Smalltalk Views. See the MacWrite document named QDViews within the same folder for a full explanation.
   b. The new MidiChannel.st goodie exercises the new MIDI driver that is built into the 0.4 interpreter. MIDI is a standard interface protocol used by sound instruments. Users who understand MIDI should be able to learn how to use the MIDI driver by looking at the MidiChannel.st goodie.
   c. The File&VolumeChgs.st goodie contains fixes to non-critical bugs in the file system. If you have problems with startup or shutdown of files or volumes or if files left open are not in the same state after a snapshot, you should file in this goodie. These fixes are included as a goodie, instead of an integrated part of the system, since we did not have time to fully test them before sending out the release.

Known Bugs in the 0.4 release:

1. If you change the state of the mouse or keyboard during a snapshot or other long disk activity, Smalltalk events may get out of sync. **WorkAround:** Don't play with the mouse or keyboard during a snapshot or other long disk activity.
2. Text strings longer than 16384 bytes have some bugs associated with them. This is why the browser currently has a 16000 character limit. Array larger than this size may have problems in general. This also affects dictionary inspectors and changes browsers whose total character length of all the dictionary's keys or method names is greater than 16384. When you scroll down to the bottom of the list, you will get a notifier window.
3. Not all the Toolbox calls work properly with the image as shipped. **WorkAround:** You can encode Toolbox calls yourself using primitive 160 as described in the 0.3 manual.
4. When you start a process that opens a file and then goes into the debugger, if you close the debug window, the file may not get closed. If you then restart this process, you may get an error stating that the file is already open once for read/write access. **WorkAround:** Whenever you have a problem with a previous instance of a file, you can do **HFSMacFileStream allInstances inspect** then inspect the instance that is causing the problem and close it yourself by executing **self close** in the bottom pane of it's inspector window.

Comments:

The 0.1 through 0.4 versions of Smalltalk released by Apple are not standard Macintosh applications. Unlike most Macintosh applications, they do not have a main event loop that calls **GetNextEvent**. This main event loop is necessary to make certain utilities work properly. This is why current versions of Smalltalk do not support desk accessories or work with standard Macintosh utilities such as Switcher™.

Smalltalk saves the bits representing underlying windows so they can instantly refresh when they again become the top window. This makes it much faster to switch from one window to another. It also uses up a fair amount of space, especially if you have only one Megabyte or RAM. If you get the **Space Is Low** warning too often from Smalltalk, you may want to turn off this caching bit feature. This should give you 40K or more bytes of extra memory. The amount it saves depends on the number of windows you have open. To turn off bit caching, execute a doIt on the statement **StardardSystemView dontCacheBits.**

Institutional License Agreement:

The Institutional License Agreement form near the back of the 0.3 manual states that the license is for Smalltalk 0.2 and/or 0.3. The same Agreement now also applies to version 0.4. However, it is unlikely that it will apply to any future versions.

Sending in Bug Report Forms:

Please use the bug report form at the back of the 0.3 manual to send reports of bugs you find (other than those listed on the previous page) to the Smalltalk Group. Please don't use the address on that form; instead, please send it to the following address:

The Smalltalk Group
Apple Computer, Inc.
MS 27-E
20525 Mariani Ave.
Cupertino, CA 95014

AppleLink: HAYNES1
UNIX: ...!apple!smalltalk

Macintosh™

Smalltalk-80™ for the Macintosh

Version 0.3

Contents

Preface

Smalltalk-80 for the Macintosh is Apple's enhanced version of the Smalltalk-80™ programming language and environment for the Macintosh™ computer, based on Xerox Smalltalk-80 Version I. It is intended primarily as a tool for learning and experimentation, but can also serve as a useful development environment in which to build prototypes of small to medium-sized applications and systems. The turnaround time for program changes is extremely rapid, allowing much faster development than with standard compiled languages.

This manual covers only those details specific to the Macintosh implementation; it does not attempt to discuss the Smalltalk-80 language and user interface. If you are not already familiar with Smalltalk-80, you can learn more about it by reading the existing literature on Smalltalk and object-oriented languages in general. A basic bibliography is included in Appendix E.

About this release

This Macintosh Plus release (version 0.3) of the Smalltalk-80 programming system is the latest of several unsupported versions of Smalltalk-80 released by Apple Computer, Inc., in the hope of allowing interested parties to get "hands-on" experience with Smalltalk. This release is intended to improve the earlier version 0.2 release of Smalltalk-80 for the Macintosh (August 1985). It generally conforms to the Xerox user interface for Smalltalk-80 but deviates in many ways from the standard Macintosh user interface.

Version 0.3 is intended to run on a Macintosh Plus with a minimum of one megabyte of memory and at least one 3.5-inch disk drive. It can also run on a Macintosh XL with one megabyte or more of memory, or on other Macintosh computers with third-party memory upgrades of one megabyte or more. (Third-party implementations must be contiguous and follow Toolbox memory organization.) If you have less than one megabyte of memory, you should get version 0.2 of Smalltalk-80 for the Macintosh, which contains a stripped-down image that will run on 512K machines.

A hard disk makes the system more usable, but it can be run from one or more double-sided 3.5-inch disks. When you are using a hard disk, the complete sources to Smalltalk-80 for the Macintosh are available on-line. Without a hard disk you can only see decompiled sources without comments.

New features of this release

New features for this Macintosh Plus version include

- □ faster painting and repainting of windows and text
- □ Hierarchical File System (HFS) support
- □ more complete Macintosh Toolbox access, including Pascal record compatibility
- □ greater compatibility with Xerox Smalltalk-80, Version II
- □ new and better text-editing facilities
- □ friendlier browser interface
- □ easier finding of hidden windows
- □ better support for change management
- □ many bug fixes

Some features of Smalltalk-80 for the Macintosh carry over from the version 0.2 release. For example:

- □ Volume management supports many file servers.
- □ Automatic spelling correction is provided for variable and message names.
- □ Multiple images can reside on the same disk.
- □ One interpreter works on all machines and uses all available memory.
- □ Up to 32,000 objects are supported.

Here are some additional details of new features. This list will be especially meaningful if you have used version 0.2 of Smalltalk-80 for the Macintosh.

- Window update is much faster, since bitmaps of underlying windows are saved. This makes the whole system faster and more usable, but it also uses more space. If the system is about to run out of space, it will turn off the fast window feature, then notify you that space is running low. To turn on the fast window feature again, choose `fast windows` from the system menu.
- The window menu, which is accessed by pressing the mouse button in a window title tab, is now also available by using the Command key with the mouse, rather than the Enter key as in version 0.2.
- You can now run Smalltalk from inside a folder under the Hierarchical File System (HFS). The image and changes files should be in the same folder. The `Smalltalk-80.sources` file may be either in that folder or in the root directory of the same volume. Once Smalltalk is running, you can access any folder on any disk, using the `choose volume` command in a file-list window (described in Chapter 2).
- If you look at the menus in a browser, you will notice some new entries. The most useful of these is `versions`. The `versions` command allows you to open a window on the current version of a method as well as previous versions still available in the changes file. If you change a method, then want to put it back the way it was before, use `versions`. You will also notice that you can now rename message categories, classes, and class categories from a menu. Rearranging or removing class and message categories is also easier.
- Version 0.3 contains many new text-editing features, described in Chapter 3.
- When you file out a class or class category, you can now also optionally file out any shared pools used by that part of the system.
- Many other improvements and bug fixes make the system faster and easier to use, such as the `reorganize` command. Most of these will be obvious as you use the system.

Relation to Xerox Smalltalk-80

Smalltalk-80 for the Macintosh is based on Xerox Smalltalk-80 Version I, and was licensed from Xerox Corporation as part of an early collaboration on Smalltalk development. Apple has made many improvements to it since that time. Xerox has also made changes and additions to its version, which it now offers for general licensing as Version II. While there are many differences between Smalltalk-80 for the Macintosh and Xerox Smalltalk-80 Version II, the Smalltalk language is still compatible and most of the kernel programming tools operate similarly.

Support

This manual should provide you with enough information to get started. To work most effectively with Smalltalk-80 for the Macintosh, you will need further documentation not provided by Apple; see Appendix E, "Bibliography of Smalltalk Literature."

Courses and training materials relating to Smalltalk-80 are available from PPI, an organization that offers general training in object-oriented programming as well as specific courses on Smalltalk-80. For further information, contact

Productivity Products International
27 Glen Road
Sandy Hook, Connecticut 06482
(203) 426-1875

Apple Computer, Inc., does not provide support for this pre-product release. If you should discover bugs in the documentation or in the system itself, your only recourse is to work around them yourself. (Even though this is an unsupported product, however, we do appreciate feedback and bug reports. Use the bug report form at the end of this manual.) If Apple later releases another version of Smalltalk-80 for the Macintosh, there is no promise of either an upgrade price or a guaranteed code migration path.

How to use this manual

Chapter 1 introduces you to Smalltalk-80 for the Macintosh and describes the installation procedures to get you started. Chapter 2 describes the user interface, with particular attention to the ways in which it varies from Xerox Version II. Chapter 3 focuses on the new text-editing facilities, and Chapter 4 on backup and change management. Chapter 5 covers enhancements relating to the Macintosh Plus Hierarchical File System (HFS). Chapter 6 discusses Macintosh-specific issues for programmers who want to use the Macintosh User Interface Toolbox from within Smalltalk.

The Appendixes provide detailed technical information for the serious programmer. Appendix A summarizes useful expressions included in the System Workspace window. Appendix B describes the Smalltalk "goodies," utility and demonstration programs distributed with the system. Appendix C discusses various useful techniques for memory management and economy. Appendix D summarizes known bugs and limitations of this release. Finally, Appendix E gives a bibliography of available literature on Smalltalk and object-oriented programming in general.

Chapter 1

Installation

You need a hard disk to run the full configuration of Smalltalk-80 for the Macintosh. This allows you to keep all the relevant files immediately available on-line. If you don't have a hard disk, you can run a more limited configuration from one or two 3.5-inch disk drives, provided that at least one of them is double-sided. This chapter gives instructions for setting up Smalltalk in either configuration, depending on the hardware you have available.

Disk contents

Smalltalk-80 for the Macintosh uses four files:

- `Smalltalk-80.interp` contains the low-level Smalltalk-80 interpreter, the Smalltalk loader, and method tables for calling the Macintosh Toolbox.
- `Smalltalk-80.sources` contains the original source text of the Smalltalk system as shipped.
- `Smalltalk-80.image` contains the compiled object code for all methods in the system, including both those supplied with the release and those that you add or modify yourself.
- `Smalltalk-80.changes` is a text file containing a record of all your actions as you work with the system.

The interpreter, image, and changes files must always be present for Smalltalk to run. The sources file is optional, and you can run the system without it—for instance, if you don't have a hard disk to keep it on. If you're connected to a network, it's also possible to access the sources file from a remote file server; see "Remote Access to System Sources," below. As you interact with Smalltalk, both the image and changes files are constantly updated. For instance, when you compile a new method with the `accept` command, its object code is written to the image file, and its source text is written to the changes file. The contents of the sources file are never changed.

When you browse a method, the system looks for the source code first in the changes file; if it isn't found there, it's taken from the sources file instead. If no sources file is present, the method's object code is automatically decompiled from the image file. This produces readable source code, but with all comments stripped out and with artificial variable names (`t1`, `t2`, and so on) for all parameters and temporary variables. If you want to see the original source code for a particular method, you can use the goodie `RetrieveSources` (described in Appendix D) to read the code directly from the 3.5-inch release disks.

The current release of the system is shipped on seven 3.5-inch Macintosh disks. For compatibility with all existing Macintosh hardware, all disks are single-sided. The image and sources files are broken up into pieces small enough to fit on the single-sided disks; a utility program, DivJoin, is included for reconstituting the complete files on a double-sided or hard disk. The contents of the release disks are as follows:

Disk Name	Filenames

Smalltalk Disk 1	`Smalltalk.changes`
	`System Folder`
	`DivJoin 1.0d9`

Smalltalk Disk 2	`2.Smalltalk.image`
	`Smalltalk.interp`
	`Goodies-Demos` folder

Smalltalk Disk 3	`1.Smalltalk.image`
Smalltalk Disk 4	`1.Smalltalk-80.sources`
Smalltalk Disk 5	`2.Smalltalk-80.sources`
Smalltalk Disk 6	`3.Smalltalk-80.sources`
Smalltalk Disk 7	`4.Smalltalk-80.sources` `Goodies-Utilities` folder

Installing Smalltalk with a hard disk

To run the full configuration of Smalltalk-80 for the Macintosh, your hard disk must be capable of holding the following files, plus about 1450K of working space (roughly 3650K total).

File size (in K)	Filename
1442	`Smalltalk-80.sources`
669.5	`Smalltalk.image`
0.5	`Smalltalk.changes`
54.5	`Smalltalk.interp`
15	`DivJoin 1.0d9` (not needed on the hard disk after initial installation)

The assorted goodies included on the release disks are optional, and total about 97K. If you have extra space on your hard disk, you might want to keep backup copies of the image and changes files, to avoid having to repeat the full installation procedure the next time you want to start from a clean copy of Smalltalk.

Here is the installation procedure:

1. Make sure the `System` file on your hard disk is version 3.2 or later and the Finder is version 5.3 or later. (These are the versions shipped with this release.)
2. Create a folder on your hard disk for your Smalltalk files. Copy the application `DivJoin 1.0d9` (on *Smalltalk Disk 1*) into this Smalltalk folder.
3. Use DivJoin to join the files `1.Smalltalk-80.sources` through `4.Smalltalk-80.sources` (on *Smalltalk Disk 4* through *Smalltalk Disk 7*), calling the joined file `Smalltalk-80.sources`. (See "Using DivJoin," below.) Place the joined file in your Smalltalk folder.
4. Copy the files `Smalltalk.changes` (on *Smalltalk Disk 1*) and `Smalltalk.interp` (on *Smalltalk Disk 2*) to your Smalltalk folder.
5. Use DivJoin to join the files `1.Smalltalk.image` (on *Smalltalk Disk 3*) and `2.Smalltalk.image` (on *Smalltalk Disk 2*), calling the joined file `Smalltalk.image`. Place the joined file in your Smalltalk folder.
6. If you wish, you can now remove `DivJoin 1.0d9` from your Smalltalk folder.
7. Move any goodies you wish to use into your Smalltalk folder. (The goodies are described in Appendix D.)
8. To start Smalltalk from the Finder, open the `Smalltalk.image` file either by double-clicking its icon or by choosing `Open` from the `File` menu. It takes about half a minute to load the image into memory and initialize the system, after which Smalltalk's startup screen is displayed. (If you have more than one version of the Smalltalk interpreter on your disk, you must indicate which one to start up. Select both `Smalltalk.image` and the interpreter you want by using the Shift key or by dragging a selection rectangle around both icons, then choose `Open` from the `File` menu or type Command-O. Notice that the interpreter and image files must be in the same folder on the disk.)

Installing Smalltalk without a hard disk

The following installation procedure assumes that you have a Macintosh Plus with one internal double-sided disk drive and no external drive. (An external drive can save you some disk swapping, but the procedure is essentially the same.) You will be creating two double-sided disks from the single-sided disks distributed in the release. Without a hard disk, you won't be able to keep the source code of the system on-line; instead of reading source text from the disk, Smalltalk will automatically decompile each method as you browse it.

To install Smalltalk on double-sided 3.5-inch disks, follow these steps:

1. Start up the system from release disk *Smalltalk Disk 1*.
2. Initialize two new double-sided disks and name them *Smalltalk System* and *Smalltalk Work*.

3. Copy the entire contents of *Smalltalk Disk 1* to *Smalltalk System*. Put away *Smalltalk Disk 1* as a backup, in case you ever need to repeat this installation procedure.
4. Restart the system from the *Smalltalk System* disk.
5. Copy the file `Smalltalk.interp` from *Smalltalk Disk 2* onto *Smalltalk Work:*
   a. Eject *Smalltalk System*.
   b. Insert *Smalltalk Work*.
   c. Eject *Smalltalk Work*.
   d. Insert *Smalltalk Disk 2*.
   e. Copy file `Smalltalk.interp` onto *Smalltalk Work*.
   f. Eject *Smalltalk Work*.
   g. Insert *Smalltalk System*.
6. Install the file `Smalltalk.image` on the *Smalltalk Work* disk:
   a. Run `DivJoin 1.0d9` from the *Smalltalk System* disk.
   b. Click the `Join` button.
   c. Eject *Smalltalk System*.
   d. Insert *Smalltalk Disk 3*.
   e. Select `1.Smalltalk.image` as the file to join and click `Open`. *Smalltalk Disk 3* will be ejected and you will be prompted to insert *Smalltalk System*. Continue to swap these two disks as prompted.
   f. When prompted for the name of the output file, eject *Smalltalk System*.
   g. Insert *Smalltalk Work* and click `Save`. Continue to insert disks as prompted.
   h. When prompted for `2.Smalltalk.image`, insert *Smalltalk Disk 2*. (You may need to use Command-Shift-1 to eject the *Smalltalk System* disk at this point.)
   i. Click `Done` when you are prompted for `3.Smalltalk.image`. (There are only two parts to be joined.) Insert the *Smalltalk Work* disk when prompted. There will be a delay while DivJoin writes the joined file to the disk.
   j. The *Smalltalk Work* disk will be ejected and you will be asked to insert *Smalltalk System*.
   k. A message will appear indicating that the join is complete. Click `OK` to continue.
   l. Exit DivJoin by clicking `Quit`.

7. You are now ready to run Smalltalk:
   a. Start up the system from the *Smalltalk System* disk.
   b. Eject *Smalltalk System*.
   c. Insert *Smalltalk Work*.
   d. Double-click on the file `Smalltalk.image`. Although you'll have to do several disk swaps to launch Smalltalk, you generally won't have to swap disks once you're running.

Remote access to system sources

Smalltalk always expects the file containing the system source code to be named `Smalltalk-80.sources`. If there is no available file by that name, it will use decompilation to display the source code for system methods. If your Macintosh is connected to a network, the system sources file can be accessed from a remote file server, provided that

- □ there is a file server on the network
- □ the server is mounted as a volume on your Macintosh
- □ there is a copy of the file named `Smalltalk-80.sources` in the root directory on the server

Remote access allows many Macintoshes without hard disks to share the same copy of the system source code over the network.

Using DivJoin

Included on *Smalltalk Disk 1* in this release is a new version of the DivJoin program that is easier and more flexible than earlier versions you may have used. DivJoin allows you to transport large files via 3.5-inch disks. It divides big files into pieces small enough to be stored on 3.5-inch disks, and later rejoins the pieces into a single file. DivJoin works only on files with a data fork and no resource fork, such as the `Smalltalk.image` file.

The new version of DivJoin, version 1.0d9, supersedes all earlier versions. (If in doubt, use the `Get Info` command to look for the version number in the file's Finder comment.) Unlike earlier versions, the new DivJoin allows you to go directly to and from multiple disks. To supplement the installation procedures given above, here are complete instructions on using the new DivJoin program for dividing and reconstituting large Macintosh files.

Running DivJoin

DivJoin begins with a dialog box with buttons labeled `Join`, `Divide`, and `Quit`. The dialog box also includes radio buttons for specifying the size of divided files. (These buttons don't apply when you're joining files.)

Whether you click `Divide` or `Join`, you're presented with a Standard File dialog box for selecting the file to be divided or the first piece of the divided file to be joined. The file lists presented by Standard File are often empty, because files that are inappropriate to the selected operation are filtered out. Only files large enough to need dividing, or named appropriately for joining (pieces starting with `1.`), appear.

Once you've selected a file, DivJoin prompts you to insert disks as needed, and displays a progress/status dialog as it runs. The progress dialog tells whether you're dividing or joining, the name of the composite file, the buffer size (all of available memory is used), and what DivJoin is doing at the moment. It also displays a `Cancel` button that you can use to cancel the divide or join operation at any time. A `Cancel` button is also included in each of the dialogs that prompt you to insert disks.

DivJoin doesn't eject the disks in both drives when a divide or join operation starts. As a result, the disk you want may already be inserted when you're prompted for it. When that happens, use Command-Shift-1 (for the internal drive) or Command-Shift-2 (for the external drive) to eject the disk, then reinsert it.

Dividing a file

Here is the procedure for dividing a large file into pieces:

1. Use the radio buttons to specify whether the file is to be divided onto single- or double-sided disks.
2. Click the `Divide` button.
3. Use the Standard File dialog to select the file to be divided. Only files large enough to need dividing are listed (those larger than a single- or double-sided disk, depending on which radio button you clicked).
4. Insert disks as prompted for succeeding pieces of the file.
5. When the operation is finished, you'll get an alert saying whether it was successfully completed or stopped prematurely (either because of an error or because you canceled).

DivJoin tries to be considerate about the disks you insert and how it formats them. If you insert a readable Macintosh disk that already has files on it, or a non-Macintosh disk, or a damaged one, you'll get an alert and have a chance to reconsider. If the disk is already formatted with the appropriate number of sides, DivJoin skips reformatting it to save time. The only mistake it cannot detect is inserting a double-sided disk in a single-sided drive.

Joining a divided file

To join the pieces of a divided file, follow these steps:

1. Click the `Join` button in the initial dialog box. The radio buttons labeled `Size of divided files` don't apply.
2. Use the Standard File dialog to select the first piece of the file to be joined. Only files with the prefix `1.` are listed, since that's how DivJoin names the first piece of divided files.
3. Use the next Standard File dialog to name the output file and to specify the destination disk, then click `Save`. DivJoin can't tell in advance whether the entire file will fit on the destination disk you select, though it does check that at least the first piece of the file will fit.
4. Once the first piece of the file has been read, DivJoin ejects the disk it's on and prompts you to insert succeeding ones. Click `Done` after reading the last piece of the file.
5. If the first piece of the file is on a hard disk, DivJoin looks for succeeding pieces in the same folder on the hard disk; it assumes it's finished when there are no more.

Limitations of DivJoin 1.0d9

DivJoin 1.0d9 suffers from the following known limitations. If you encounter any others, please report them as bugs.

- A file's resource fork, if any, is not included when you divide the file.
- No information about divided files is written with them; data corruption errors, or your omission of the last file(s), cannot be detected.
- I/O errors are displayed numerically. (See the result codes in *Inside Macintosh,* Volume III, pp. 205–209).
- If a divide or join operation fails, the incomplete files are not deleted.
- Divided pieces of a file retain the type and creator of the composite file, so they can still be opened by double-clicking from the Finder. The file piece probably won't make much sense to the application that created it.

- ☐ Unlike the original DivJoin, version 1.0d9 always divides to 3.5-inch disks. If you want the divided pieces of a file on the same hard disk, you must copy them there from the 3.5-inch disks after leaving DivJoin.
- ☐ The list of files for a divide operation sometimes omits files large enough to need dividing. For instance, a 781K file will not appear in the dialog box, even though it's too large to fit on a double-sided disk (which has 779K free after initialization). The only workaround at present is to divide to single-sided instead of double-sided disks.

Chapter 2

User Interface

This chapter describes the user interface of Smalltalk-80 for the Macintosh, and in particular the ways in which it varies from Xerox Version II. For a full discussion of the Xerox Smalltalk-80 user interface, see ***Smalltalk 80: The Interactive** Programming Environment*, by Adele Goldberg (listed in Appendix E, "Bibliography of Smalltalk Literature").

Scroll bars

Just as in standard Macintosh applications, the full contents of a window or pane are usually not visible at any given time. Scroll bars are provided to select what portion will be displayed. These are similar in spirit to the standard Macintosh scroll bars, but there are some differences in operation.

To operate a scroll bar, move the cursor into the window or pane you wish to scroll. After a short pause, the scroll bar will appear at the left side of the window. The gray *marker* inside the scroll bar indicates the position of the currently visible portion relative to the overall contents of the window. Unlike the standard Macintosh scroll box, however, the height of the marker within the whole scroll bar corresponds to the proportion of the window's contents that are currently visible.

To scroll, move the cursor to the left or right of the marker. It will change its shape to an arrow showing the direction in which the contents of the window will scroll when you click the mouse: up when the cursor is to the right of the marker, down when it's to the left. The vertical position of the cursor within the scroll bar controls the distance the window will scroll. Scrolling up moves the line of text directly opposite the cursor to the top of the window; scrolling down moves the current top line down to where the cursor is. If you press and hold down the mouse button, the window will scroll continuously until you release the button or until it reaches the top or bottom of the window's contents.

If you move the cursor into the gray marker itself, it changes into a small black dot. By pressing and holding down the mouse button at this point, you can drag the marker directly to the desired position, just as in a standard Macintosh scroll bar. As you drag the marker, a dotted outline shows its original position, so that if you change your mind and want to cancel the scroll you can move it back to where it started before releasing the button. Directly above or below the marker, the cursor changes to a right-pointing arrow. Clicking at this point will jump the marker directly to the indicated position and scroll the contents of the window accordingly.

Pop-up menus

Instead of the familiar Macintosh pull-down menus that run across a menu bar at the top of the screen, Smalltalk uses *pop-up menus* that appear right at the current cursor position when you press the mouse button. The particular menu you get depends on the area of the screen in which you press the button; you can also control the choice of a menu by holding down certain keys on the keyboard in combination with the mouse button.

The Xerox mouse buttons

The original Xerox implementation of Smalltalk-80 was designed for a three-button mouse. The buttons, designated *red, yellow,* and *blue,* have the following functions:

Button	Function
red	Selects information within a window
yellow	Displays a menu for operating on the contents of a window
blue	Displays a menu for operating on the window itself

The Xerox three-button user interface is fully described in the Goldberg book, *Smalltalk-80: The Interactive Programming Environment.* In Smalltalk-80 for the Macintosh, other conventions are used to adapt this interface to the one-button Macintosh mouse.

In general, the Macintosh mouse button corresponds to the red button on the Xerox mouse. You use this button to select text or list items within a window, to position windows when moving or resizing them, and so forth. Holding down the Option key while pressing the mouse button makes it behave like the Xerox yellow button; the Command key transforms it into the Xerox blue button. In addition, moving the cursor into certain areas of the screen makes it behave like the yellow or blue button instead of the red one:

- □ Near the right edge of any scroll bar, the mouse button works like the Xerox yellow button, displaying the pop-up menu specific to that window or pane. To signal this behavior, the cursor changes to look like a little menu.
- □ In a window's title tab, the mouse button works like the Xerox blue button, displaying a *window menu* for manipulating the window itself (moving, resizing, closing, and so forth) rather than its contents.
- □ In the gray background area outside of any window, the mouse button works like the Xerox yellow button, displaying a *system menu* of global commands applying to the system as a whole.

Notice that these conventions eliminate the need for the Option and Command keys in conjunction with the mouse: by moving into the proper areas of the screen, you can invoke all the functions of the Xerox red, yellow, and blue buttons via the single Macintosh mouse button.

The system menu

The *system menu* consists of global commands that apply to the system as a whole. To display it, press the mouse button anywhere in the gray background area of the screen, outside of all windows. The system menu contains the following commands:

Command	Action
`restore display`	Restores correct screen appearance; repaints all windows
`find window`	Brings a designated window to front of screen
`fast windows`	Enables fast redrawing of windows (see "New Features," below)
`open project`	Opens a new project window
`open browser`	Opens a new system browser
`open workspace`	Opens a new workspace window
`open file list`	Opens a new file-list window
`open transcript`	Opens a new transcript window
`eject disk`	Ejects a disk
`exit project`	Exits to "parent" of current project
`snapshot`	Saves current state of system to disk
`quit`	Exits from Smalltalk

The window menu

The *window menu* contains commands for manipulating a window itself, rather than its contents. To display this menu, either press the mouse button in the window's title tab, or press it anywhere within the window while holding down the Command key. The window menu contains the following commands:

Command	Action
`label`	Edits title in window's title tab
`move`	Changes window's position but not its size
`frame`	Changes both window's position and size, or restores a collapsed window to its original position and size
`collapse`	Shrinks window to just a title tab
`close`	Destroys window permanently

Other menus

In addition to the system and window menus, each type of window or pane has its own specialized menu for manipulating or operating on its contents. To display this menu, either press the mouse button near the right edge of the scroll bar (when the cursor appears as a picture of a little menu), or press it anywhere within the window or pane while holding down the Option key. The contents of these specialized menus vary, depending on what's appropriate for a particular window or pane. For instance, a workspace window presents the standard text-editing menu described in Chapter 3.

Menu differences from Xerox Smalltalk-80

The menus in this version of Smalltalk-80 for the Macintosh differ from those in Xerox Version II Smalltalk in the following ways:

Menu	Xerox command	Macintosh command
System	`project`	`open project`
	`browser`	`open browser`
	`workspace`	`open workspace`
	`file list`	`open file list`
	`transcript`	`open transcript`
	`save`	`snapshot`
	(not supported)	`find window`
	(not supported)	`fast windows`
	(not supported)	`eject`
Browser, class categories	`spawn`	`browse`
	`add category`	`add item`
	`edit all`	`reorganize`

Menu (continued)	**Xerox command**	**Macintosh command**
Browser, class names	`spawn`	`browse`
	`protocols`	`reorganize` (moved to message categories)
	`spawn hierarchy`	(not supported)
Browser, message categories	`spawn`	`browse`
	`add protocol`	`add item`
Browser, message selectors	`spawn`	`browse`
	`move`	(not supported)
	(not supported)	`versions`
Dictionary inspector	`add field`	`add key`

The goodie `Version II Menus` (described in Appendix B) will make the menus more compatible with those in Xerox Version II. There are also significant differences in the pop-up menu for spelling correction; see Chapter 3 for further discussion.

File lists

This release of Smalltalk-80 for the Macintosh runs under the Hierarchical File System (HFS) of the Macintosh Plus, but does not yet have an iconic interface to represent files and folders. In place of the Macintosh Standard File dialogs, it uses an enhanced version of the Smalltalk file-list browser to access the files in an HFS folder hierarchy.

To open a file list on your screen, use the `open file list` command on the system menu. The `choose volume` command in the top pane of the file-list window displays a menu of volumes and folders currently known to the system. Names followed by three dots (...) denote a volume or folder that, in turn, contains subfolders of its own. When you choose one of these, you get a new menu showing the subfolders it contains. Choosing the same volume or folder twice in a row gives you a list of its files in the second pane of the file-list window. When you select any of these files, its contents appear in the bottom pane. You can then read the file's contents or edit them and write them back out with the `put` command. The `get` command cancels any editing you may have done and reverts to the file's original contents as read from the disk.

Each time you use `choose volume`, the system will remember all the volumes or folders you select along the way and will display them in the top-level volume menu the next time you use the command. This saves you the trouble of navigating back through the hierarchy the next time you want to access the same folder. However, such "remembered" folders are forgotten when you quit and restart the system, unless they are referred to by some existing object in your saved image (such as an open file-list window).

In addition to remembered folders, you may see the following names on your volume menu:

Name	Meaning
`Internal`	Internal 3.5-inch disk drive
`External`	External 3.5-inch disk drive
`Device3`	Hard disk
`MacDefault`	Volume or folder from which Smalltalk was started up

The file list always comes up initially displaying the last of these (the folder from which you originally started up Smalltalk from the Finder).

New features

Version 0.3 of Smalltalk-80 for the Macintosh includes a number of new user interface features. These fall into the following categories:

- □ faster drawing and manipulation of windows on the screen
- □ enhancements to the browser
- □ new text-editing features
- □ improved change management

The first two categories are covered here; text editing is discussed in Chapter 3 and change management in Chapter 4.

Fast windows

Probably the most important change in this release is that Smalltalk now preserves the images of obscured windows and can therefore refresh them much more quickly. To change back and forth from one browser to another, which used to take about nine seconds, now takes about three. In addition, the screen is now properly updated and doesn't show the ugly gray "holes" characteristic of earlier releases.

There are a couple of things to know about the new window management scheme. One is that the extra stored images consume memory space, and it is possible to exhaust the available space by creating too many large windows. The system will attempt to deal with this problem as gracefully as possible. If Smalltalk runs out of memory, it will first discard all its saved window images and revert to the old slower method of window management. It will also put up a notification window informing you that it has done this. After you figure out what's wrong and make more memory available, you can choose the `fast windows` command from the system menu to re-enable fast window management.

Executing (via the `doIt` or `printIt` command) a method that draws graphics on top of the current windows can cause confusing effects on the frontmost window. In such cases, you can use the Shift key together with the `doIt` or `printIt` command; this saves the original screen image before executing the graphics method. After execution is complete, the resulting graphics will remain on the screen until you click the mouse button, at which point the entire screen will be redrawn, restoring all windows to their proper appearance.

In connection with the new window management scheme, it is now possible to install graphic forms other than the default gray pattern as the screen background. For example, executing the statements

```
QDPen new mandala: 30 diameter: 640.
ScheduledControllers backgroundForm:
    (Form fromDisplay: Display boundingBox).
```

will draw a geometric pattern on the screen, copy that pattern to a form, and then install the form as the screen background. The background form consumes a certain space overhead, however: about 20K for the 512-by-342 Macintosh screen. To revert to the dull but space-saving gray pattern, execute

```
ScheduledControllers backgroundForm: (InfiniteForm with: Form Gray).
```

Browser enhancements

The new release includes the following enhancements to the system browser:

- A new `versions` command is available in the message selectors pane. This command opens a new browser showing all available prior versions of a selected method. You can revert to an earlier version of the method by selecting the desired version and choosing the `accept` command. This can be especially useful for rescinding unsuccessful changes you may have made.
- New commands in the class and message categories panes make it easier to add and rename categories. The `rename` command allows you to change the name of the currently selected category; `add Item` adds a new one. The new category is inserted just before the one currently selected, if any; if no category is selected, the new one is added at the end of the list.
- The `rename` command in the class names pane allows you to change the name of an existing class. A new browser is opened showing all methods that refer to the class by name, so you can update them to the new name.

- When you browse all methods that refer to some object or message (as with the `senders` command in the message selectors pane), each method appears with the first reference to the selected name automatically highlighted. This helps you locate the reference visually, and makes it easy to replace it with a different name or selector (as with `paste` or `again`).
- If you make changes in a browser that have not yet been accepted, and then attempt any operation that would lose those changes, Smalltalk now highlights the affected pane and asks you whether to accept the changes before proceeding with the requested operation. Many operations which were awkward or confusing in the previous release have thus become simpler or clearer.

Chapter 3

Text Editing

Smalltalk-80's text-editing facilities are based on a cut-and-paste model similar to that used in other Macintosh applications. This implementation, Smalltalk-80 for the Macintosh, includes a number of additional features and modifications to make it even closer to the standard Macintosh model. These include, for example, using the Shift key to extend a selection and the keyboard combinations Command-X, Command-C, Command-V, and Command-Z for the standard editing operations `cut`, `copy`, `paste`, and `undo`.

However, there are still some differences between Smalltalk's editing conventions and those of the standard Macintosh user interface. In Smalltalk, for example, you can type text into a window only when that window is active and contains the cursor. When the cursor is outside the active window (or pane), anything you type on the keyboard is saved for later insertion. As soon as you move the cursor back into the window, the text you typed will be inserted; but if instead you activate a different window by clicking inside it, the saved text will be inserted in that window instead.

Another difference is the interpretation of double mouse clicks. Double-clicking within a word selects the word, just as under standard Macintosh conventions. In Smalltalk, however, double-clicking at the beginning or end of a line of text selects the entire line, and double-clicking just inside one of a pair of matched punctuation marks (such as parentheses, quotation marks, or square brackets) selects everything between the paired marks. (This feature is particularly useful in program text, for finding balanced parentheses and brackets.) For a complete description of the standard Smalltalk-80 editing features, see the Goldberg book, *Smalltalk-80: The Interactive Programming Environment.*

The editing menu

The standard editing menu offered by workspace windows (and in modified form by other types of window as well) includes the following commands:

Command	Action
`again`	Repeat last editing operation
`undo`	Undo last editing operation
`copy`	Copy selected text to scrap
`cut`	Cut selected text from window to scrap
`paste`	Paste scrap over current selection or at insertion point
`doIt`	Execute selected text as Smalltalk code
`printIt`	Execute selected text as Smalltalk code and display result
`accept`	Incorporate editing changes permanently
`cancel`	Cancel all changes since last `accept`

Typing from the keyboard doesn't disturb the contents of the cut-and-paste scrap buffer, which continues to hold the last text explicitly cut or copied from any window.

Improved undo command

The `undo` command has been overhauled for more Macintosh-like behavior. It now faithfully restores the text and selection to their condition before the last editing operation. All editing operations can now be undone (or redone by undoing the `undo`), even if you have since moved the selection. As a result, you can no longer use `undo` as a form of repeated `paste`. (Exception: When you are in a paragraph other than the one where you performed the original operation and there is no other way to get back the contents of the `undo` buffer, you can still get it back with `undo`.) The behavior of `undo` has been changed in the following ways:

- Undoing a `cut` or `copy` restores the scrap buffer to its previous contents. Thus if you cut some text and then accidentally choose `copy` instead of `paste`, you can undo the `copy` and then complete the original `paste`.
- Any sequence of typing and backspacing keys, however mixed, is treated as a single operation. An `undo` will restore everything that was deleted, and another `undo` will redo the original operation perfectly.
- Undoing or redoing a `cancel` faithfully restores the earlier state. (Please forgive the excessive screen activity that accompanies this operation.)

- Many more operations are now transparent to undo; that is, the previous operation can still be undone and redone. Besides scrolling, selecting, `accept`, and `doIt`, these now include
  - `cut` or `copy` of an empty selection
  - `again` when nothing was found
  - leaving the window or pane and returning without doing any further editing in other views

Extended again command

The `again` command has been extended in the following ways:

- Besides repeating the last `paste` or overtype at the next occurrence of the same original text, `again` can now also repeat other operations that delete or replace text—in particular, `cut`.
- The `again` command no longer has any effect on the cut-and-paste scrap buffer. If you do some typing followed by `again`, and then want to paste the same text somewhere else, use `copy` to get the selected text into the scrap buffer for pasting.
- Any sequence of typing and backspacing keys, however mixed, is treated as a single operation and can be repeated with `again`. (In previous versions of the system, you couldn't use `again` after a type-in if you had backspaced to make a correction.) If you backspace past the beginning of the original selection, the characters you backspace over will all be included in the search string.
- The `again` command repeats the last replacement in any window or pane, not just the one that is currently active. (This currently doesn't work right in "Senders of ..." windows.)
- If you hold down the Shift key while choosing `again`, it will replace every occurrence of the search string to the end of the text, not just the next occurrence. Undoing a Shift-`again` restores the text and selection to their exact prior state (except for font), and you can redo.
- Undoing an unshifted `again` just restores the text in the one place it was changed and leaves it highlighted. Redoing the `again` at this point (by undoing the `undo`) will repeat the replacement in that one place; explicitly choosing another `again` will leave it as is and instead replace the *next* occurrence of the original search text. (This is because the `again` command always starts its search at the end of the current selection.)
- In previous versions, if you wanted to search for something, you had to see an example of it, select the example, choose `copy`, then choose `again`. In this version, if you can't see an example of the text you want to search for, you can place the insertion caret anywhere, type the search text, and choose `again`. The example you typed will automatically be deleted before beginning the search. The typing and deletion of the search text are transparent to the `cancel` command;

that is, if you've made no other changes to the text, you need not cancel before closing the window or browsing to a different method.

- If the last command was `again` (or an undo of an `again`), the same search and replacement strings are used as were used the last time. In the past, if you searched for text with `again` and then did some editing, you lost the ability to search for the next occurrence of the same text. You can now use the Command-S (`same`) command (see "New Keyboard Commands," below) to continue the most recent `again` instead of repeating the intervening edit.

New keyboard commands

The following new editing commands are available via Command-key combinations on the keyboard. (None of these commands except undo is available on a menu, but `exchange` will work correctly if you add it to the editing menu yourself.)

Keystroke	Command	Action
Command-Z	undo	Same as menu command undo
Command-S	same	Repeat most recent `again`
Command-L	left	Shift selected lines left
Command-R	right	Shift selected lines right
Command-D	duplicate	Paste current selection over previous selection
Command-E	exchange	Exchange current selection with previous selection
Command-Q	query	Complete symbol preceding caret
Command-A	advance	Advance caret to next token

Same (Command-S)

Command-S (same) repeats the most recent search or replacement you performed with the menu command `again`. The new search begins at the end of the current selection. This allows you to do other editing in between `again` commands, then continue to search for more occurrences of the same text. This operation is transparent to undo and doesn't force a `cancel`.

Shift Left (Command-L) and Shift Right (Command-R)

Command-L and Command-R shift text left and right by inserting and deleting tab characters at the beginning of each line. (A line of text is considered to be any sequence of characters delimited by carriage returns or by the beginning or end of the text.) These commands operate on every line that includes at least one visible character of the current selection; lines containing only invisible formatting characters such as spaces and tabs are not affected. If the selection is an insertion caret, only the single line containing it is shifted.

Lines that don't begin with a tab character cannot be shifted left. If the selection includes any such nonblank lines, Command-L will just flash the window and leave all selected lines unchanged.

Duplicate (Command-D)

Command-D (duplicate) pastes the current selection over the previous selection, but without affecting the scrap buffer. You can use this operation to copy existing text from somewhere else while typing from the keyboard. It's also useful for copying temporary variable names from the body of a method to the declaration line at the beginning.

To be considered a separate selection for purposes of Command-D, the current selection must not overlap the previous one (though it may touch it at either end). If the previous selection was an insertion caret, the new one must not enclose it or touch it at either end. Command-D leaves the caret positioned at the end of the duplicated text (that is, in the neighborhood of the *previous* selection). To select the duplicated text, use Command-~ (see "Other Keyboard Conventions," below).

Exchange (Command-E)

Command-E (exchange) exchanges the current selection with the previous selection, without affecting the scrap buffer. Again, to be considered separate, the two selections must not overlap. Whatever text was selected before the exchange remains selected afterward, but in its new location (where the previous selection was originally). If both selections were nonempty, another Command-E is the same as an undo unless you have changed the selection in between.

One of the many uses for this command is to exchange the true and false branches of a conditional: double-click inside one of the brackets enclosing the first branch, then inside one of the brackets of the second, and type Command-E. Other common uses are to exchange two statements, two arguments of a message, or the left and right sides of a simple assignment.

Either the current or previous selection, but not both, may be empty (just an insertion caret). In this case, Command-E performs a move instead of an exchange: the nonempty selection is moved to the location marked by the empty one. This operation is useful in the same situations as Command-D (above), but when you want to move the selected text instead of copying it. A caret is left at the end of the moved text; if you want to select the text, use Command-~ (see "Other Keyboard Conventions," below). After a move, another Command-E is not the same as an undo; it just flashes the window and does nothing.

Query (Command-Q)

Command-Q (query) saves typing by completing a partially typed message selector. The current selection must be empty (an insertion caret) and must be preceded immediately by an identifier, called the ***hint***. Smalltalk searches its symbol table for a selector that completes the hint, and replaces the hint with this selector, called the *offering*. Notice that the symbol table is searched, not the method dictionaries. Only symbols that could be selectors and that begin with a lowercase letter are offered. The case (upper or lower) of characters in the hint does not matter, but spelling does.

If the offering is a single-keyword selector, it is followed by a space and then the caret. For example, if you type `desel` and then Command-Q, Smalltalk will display

```
deselect: ^
```

You can then proceed to type the argument following the selector. If the offering is a multi-keyword selector, two spaces are inserted between each pair of keywords, and the caret is placed after the first keyword, allowing you to type in the first argument without touching the mouse. For example, if you type `detec` followed by Command-Q, Smalltalk will display

```
detect: ^ ifNone:
```

After typing the argument, you can use Command-A (see below) to advance to the next argument.

If the offering displayed is not the one you want, type Command-Q again immediately. (The previous offering must still be displayed, and you must not have done any intervening editing.) Smalltalk will then search for another offering and substitute it for the first one. For example, if you type another Command-Q in the example above, Smalltalk-80 will display

```
detect: ^
```

When no further matches can be found, Command-Q will restore the original hint (before the first Command-Q) and flash the window.

If you choose undo when an offering is displayed and no further editing has intervened, the original hint will be restored. Another undo (that is, a redo) will restore the state before the first undo, and you can continue the search.

If you type Command-Q and then click the mouse elsewhere in the text while an offering is displayed, another Command-Q will restore the previous selection and continue the search with the same hint. If only the original hint is displayed, the identifier before the new position of the caret becomes a new hint and a new search begins.

Command-Q is useful when you can't remember the spelling or capitalization of a long message selector, or when you can't remember all its keywords, or when you just don't want to type a lot. It is particularly useful for long messages to the special object `Mac` (see Chapter 6), especially if you have taken the trouble to get their true keywords into the symbol table instead of a lot of `with:`'s. You can also use it when you're inventing a new message name and you want to be sure it *is* new, or conversely, if you want to try to pick a name that has been used before.

Type as many characters of the name as you can remember, then type Command-Q. Usually you'll type the first keyword, including the colon; but if you wish, you can type fewer or more characters. All of the following will be completed to the selector `detect:ifNone:`

```
detect:
```

```
deT
```

```
detect:ifn
```

The more you type, the more ambiguity you will remove and the fewer spurious answers you'll receive. If you find yourself working through a long list of matches, choose undo (or type Command-Z), add a few more characters to the hint, and type Command-Q again.

Advance (Command-A)

Command-A (advance) advances the caret to follow the next occurrence of the characters colon-space. You can use it after a Command-Q to advance to successive argument positions within a multi-keyword selector. For example, after Command-Q gives you the offering

```
detect: ^ ifNone:
```

type the first argument and then use Command-A to obtain

```
detect: firstArg ifNone: ^
```

Other keyboard conventions

Here are some other useful keyboard conventions:

Keystroke	Meaning
Command-`	Select last text typed in, pasted, duplicated, etc. (like Escape key in other Smalltalk systems)
Command-~	Same as Command-` above
Command-_	Add or remove parentheses around selection (like Command-(in other Smalltalk systems; that combination is unavailable on the Macintosh because Command-Shift-9 is reserved for a different purpose)
Shift-6	Up arrow (for returning method result)
Shift-minus	Left arrow (for assignment)
Command-period	Interrupt current operation immediately (like Control-C in other Smalltalk systems)

Spelling correction

Spelling correction is automatic rather than voluntary in this version of Smalltalk-80 for the Macintosh, and you have the opportunity to choose among the closest matches. When you accept a method or execute an expression with `doIt` or `printIt`, Smalltalk compiles the code you've selected or accepted. When the compiler encounters a symbol it doesn't recognize, it presents a menu of suggested alternatives, in case you've made a typing or spelling error. If you choose one of the alternatives, it will be substituted for the original misspelled version and compilation will proceed.

If the misspelled symbol is a message selector, the menu includes, in addition to any proposed alternative spellings,

- the text you originally typed (in case it is correct and you just haven't yet defined the method in question)
- `try harder` (widen the search for possible alternatives; the original correction algorithm favors symbols that begin with the same letters)
- `cancel` (if you see your spelling error and want to fix it yourself)

❖ *Note:* Many valid Toolbox calls (see Chapter 6) are unknown to the compiler, so it may well be appropriate for you to accept the unrecognized selector as is.

If the misspelled symbol is a variable name beginning with a lowercase letter, the `try harder` command will be replaced on the menu by `declare temp`, to add this name to the method's list of temporary variables and proceed with the compilation. If the name begins with an uppercase letter, the command will be `declare Class Variable` instead.

Miscellaneous

The text editor in version 0.3 of Smalltalk-80 for the Macintosh also includes the following miscellaneous features and improvements:

- When the selection is empty (an insertion caret), the `cut` and `copy` commands have no effect. They just flash the window, and don't change the contents of the scrap buffer. They also don't force a `cancel` if you've done no other editing.
- When the selection is nonempty, the command sequences `copy`, `copy-undo`, and `copy-again` change the scrap buffer but don't force a `cancel`.
- Command-V (`paste`) may be typed ahead without flushing previously typed characters. This is useful when you want to type a few characters before pasting, or when you want to paste the same text in several places. The `paste` and the typing on either side of it are considered separate commands for purposes of `undo`.
- In previous versions of the system, if you typed something and then clicked the mouse before all the typed characters had been processed, the editor gave priority to the mouse, made a new selection, and inserted the typeahead there. This version gives priority to the typeahead, so it will be inserted in the right place most of the time.
- The scroll bar marker (scroll box) no longer blinks after every edit and burst of typing. Typing and backspacing now adjust the marker if necessary.
- If you try to invoke the `format` command (without the Shift key down) when you have already made some edits to a method, the window flashes. You should accept or cancel before trying again. (It used to format the original text, causing your edits to vanish.)

Chapter 4

Saving Your Work

As in any computer system, it is vitally important to back up all the work you do in Smalltalk. Keep in mind, though, that because Smalltalk (unlike other programming environments) makes no firm distinction between system and application code, it's possible to modify the system itself in such a way as to leave it unusable. It is thus equally important to make sure you have a healthy, working version of the system to fall back on in case your modifications should prove catastrophic. This chapter discusses a variety of tools that Smalltalk offers to aid you in saving your work and avoiding disaster.

Images and snapshots

Your image file and changes file (see Chapter 1) together define the complete state of your Smalltalk system. The most straightforward way to back up your work is simply to save copies of these two files on external media. (You might even want to keep a historical archive of different versions of the files through their various incarnations.) You can name the files anything you like, provided that the two names end in `.image` and `.changes` and are otherwise identical: for example, `MySmalltalk-9/26.image` and `MySmalltalk-9/26.changes`. If the files grow too large to keep on a single disk, you can use the DivJoin utility (see Chapter 1) to divide them for backup.

As described in the next section, the changes file is maintained for you automatically by the Smalltalk system. Maintaining the image file is your responsibility. The `snapshot` command on the system menu saves a new version of the image file, recording the exact state of your working environment, down to the arrangement of windows on the screen and the current selection in the active window. The new snapshot is written out to the same image file you used when you started up Smalltalk from the Finder.

Each time you leave Smalltalk with the `quit` command, you're asked whether you want to save your changes since the last snapshot. If you answer `yes`, a snapshot of your current state is saved to your image file before quitting, so that your next working session will begin exactly where this one left off. If you answer `no`, your working state at the start of the next session will revert to the previous snapshot.

In general, you should make a snapshot before undertaking any critical change that might damage or compromise your system. However, don't forget that the snapshot will replace the previous (presumably "safe") version of the image file that you already have on your disk. If you think you'll be making a snapshot, it's a good idea to make a backup copy of your existing image and changes files in the Finder, before starting up Smalltalk.

If you've made some changes that you want to save in a snapshot, but you forgot to back up your files in advance, all is not lost. First file out the changes you wish to save (see "File-Out and File-In," below). Then quit to the Finder without saving changes, make backup copies of your image and changes files, and restart Smalltalk. Now you can file in your changes and safely make a snapshot.

Changes and crash recovery

As you work with Smalltalk, the changes file keeps a log of everything you do. Each time you compile a method with the `accept` command or execute an expression with `doIt` or `printIt`, a copy of the code is appended to the changes file in Smalltalk's standard file-out format (see "File-Out and File-In," below). This happens automatically, even if you don't explicitly save a snapshot or file out your changes. If there is an unexpected system crash, you thus have a complete record in the changes file that you can use to reconstruct your work and restore the state of your system.

If you crash and lose your work, select and execute the statement

```
Smalltalk recover: 5000.
```

in your System Workspace window. This will copy the last 5000 characters of the changes file to a temporary file named `st80.recent`, and open an editing window on that file. (You can vary the number of characters as needed; the only limitation is that imposed by the file window, currently 16,000 characters.) You can now recreate your lost work by selecting the relevant information in this window and filing it back in. The file window's menu command `fileItIn` operates only on the current selection, not the entire file. In the case of method definitions, the selection must include the `methodsFor:` line preceding the definition, as well as the exclamation points that delimit it at the end. Another way to examine and selectively file in code from the changes file is with a change-list browser, described below under "File-Out and File-In."

The changes file grows every time you run Smalltalk. If you redefine a method twenty times, there will be twenty copies of it in the changes file. This can be useful if you want to cancel your recent changes and revert to an earlier version of the method; the `versions` command in the browser's message selector pane allows you to browse all the earlier versions of a method. Even if you don't save changes when you quit Smalltalk, your changes file will have grown during the session. Eventually, you'll find it necessary to condense the changes file: that is, to write a new copy containing only the code accessible from the current image. To do this, execute the statement

```
Smalltalk condenseChanges.
```

in your System Workspace window. The condensing process can take time (ten minutes or more), depending on the size of the changes file. It's a good idea to make backup copies of your image and changes files before condensing and take a snapshot immediately afterward.

If you don't have a hard disk, you must pay particular attention to the amount of free space on the disk you keep your changes on. The condensing process itself requires additional space for the condensed copy, so you should ideally condense the changes file before it becomes larger than the space remaining on the disk. If this becomes too restrictive (and if you're very brave), you can use the goodie `DropChanges`, described in Appendix D. This reclaims all disk space occupied by the changes file, but at the expense of losing all the source code the file contains. Future browsing of the affected methods will present only a decompiled version of the code; all variable names and comments will be lost. Subsequent changes will continue to be logged in the changes file, allowing full source browsing until you drop changes again. If you must resort to this solution, you should probably first file out any code you care about, to retain a copy of the code with comments (see "File-Out and File-In," below).

File-out and file-in

You can save parts of your Smalltalk system to external files in *file-out format;* this provides a more compact way of saving selected changes than `snapshot`, which saves your entire image and all changes it contains. The `fileOut` commands in the various panes of a browser window write all the code of a selected class category, class, method category, or method to a file. This code can later be read back in from a file-list or file window with the `fileIn` or `fileItIn` commands. File-out and file-in operations are also available by executing certain Smalltalk expressions; see your System Workspace window and Appendix A for examples. A detailed description of the file-out format is given in the article "The Smalltalk-80 Code File Format," by Glenn Krasner, in Krasner's book *Smalltalk-80: Bits of History, Words of Advice* (see Appendix E, "Bibliography of Smalltalk Literature").

File-out format allows code to be moved from one Smalltalk image to another and to be read or edited with any Macintosh text editor, such as the Macintosh Development System (MDS) Edit program or the Macintosh Programmer's Workshop (MPW) editor. You can also edit such files in MacWrite™ or Microsoft Word™, if you take care not to disturb the file-out format conventions. (However, you must remember to save the files in text-only form so they can later be read back into Smalltalk.)

Note that browsers do not immediately reflect changes made in other browsers or in file-ins. It may be necessary to reselect an item in order to display the current version. The `update` command in the browser's class categories pane causes it to display new system categories created by another browser or by a file-in. This command is the most reliable way to make sure the browser reflects the current state of the system.

The change set and projects

In addition to the changes file, Smalltalk provides another, completely independent mechanism for recording your changes. Whereas the changes file is a sequential log that records the text of all methods compiled and expressions executed, the *change set* is an unordered collection telling which classes and methods have been changed (added, deleted, modified, renamed, or reorganized). A statement of the form

```
(FileStream newFileNamed: 'MyChanges-9/26.st') fileOutChanges.
```

will file out all classes and methods in the change set to the named file. You can then empty the change set, to begin collecting new changes, with the statement

```
Smalltalk noChanges.
```

Change sets combine with Smalltalk's *project* mechanism to provide a convenient way of organizing and modularizing your file-outs. Each project window you open (via the `open project` command on the system menu) represents a separate working environment, with its own desktop, its own set of windows, and its own independent change set. By using a separate project window for each programming task, you can keep your changes separate and file each out in its own file.

Here is a step-by-step procedure for using projects and change sets for modular program development:

1. Start with a relatively stable version of the system and back it up on a disk before you begin.
2. Start Smalltalk.
3. Open a new project window and enter it to start a new change set.
4. Isolate exactly what makes up the programming change (new feature or bug fix) you wish to make.
5. Code the change, commenting extensively for future reference.
6. File out the change set for this project, using the Smalltalk statement shown above.

7. Quit Smalltalk and return to the Finder.
8. If you wish, edit the file-out to add comments at the top describing its contents and purpose.
9. Restart Smalltalk to test your file-out, using the backup copy as the original image that you made in step 1 above.
10. File in the change.
11. Test the change.
12. Fix if necessary and file out the updated change.
13. Make a snapshot to incorporate the change into your Smalltalk image.

You can now give a copy of your file-out to someone else or file it into a different image. Use the `select conflicts` command in a change-list browser (see next section) to avoid code conflicts when porting a change from one image to another.

Change-list browsers

This version of the system includes a new type of browser, called a *change list.* It allows you to browse files in file-out format, by presenting one list item for each object in the file. This provides a way to examine file-out files that are too long for an ordinary file window, which is limited to 16,000 characters or less. The browser's list pane permits multiple selections, allowing selected parts of the file to be filed in or copied to another file.

To create a change-list browser for a file, execute a statement of the form

```
ChangeList browseFile: 'MyChanges-9/26.st'.
```

One use for a change list is to browse the system changes file itself. The statement

```
ChangeList browseRecentLog.
```

in the System Workspace window opens a change list for browsing all changes since the last snapshot. This is useful for recovering changes after a fatal error or power failure. After restarting Smalltalk, you can open a browser and file in all your unsaved changes (or, if you wish, only selected ones).

It's also possible to browse farther back in the changes file than the last snapshot. To do this, execute a statement like

```
ChangeList browseRecent: 10000.
```

giving a large enough character count (say 10,000 or 20,000) to reach as far back as you want to go. This can be useful for selecting just certain desired changes from among others that are not of interest. You can then file out just those changes to a separate file and give them to someone else to use.

The `select conflicts` command in the change-list browser's list pane helps you detect code incompatibilities when filing in changes from several different files (perhaps developed independently by different people). After filing in each set of changes, you can open a change list on the next and choose `select conflicts`. This command selects every method definition in the given file that conflicts with one already included in the current system change set. You can then file in all but the conflicting methods, examine the conflicting ones, and try to resolve the conflicts in whatever way is appropriate.

Chapter 5

The Hierarchical File System

This chapter describes the new features in this release of Smalltalk-80 for the Macintosh that let it work with the Macintosh Plus Hierarchical File System (HFS).

Volumes and folders

Macintosh volumes are represented by Smalltalk objects of class `HFSMacVolume`, a subclass of `FileDirectory`. (Throughout this chapter, the word *volume* is understood to refer either to a physical volume such as a 3.5-inch or hard disk, or to a folder within the Hierarchical File System.) Any file name may be preceded by an optional *volume prefix,* separated from the file name by a colon:

```
volumePrefix:fileName
```

If the volume prefix is omitted, the file is understood to reside on the *default volume.* Initially the default volume is the one containing the image file from which Smalltalk was started up; you can change it to a different volume by sending a message to class `FileDirectory` of the form

```
FileDirectory default: 'MyVolume'.
```

If you want no default volume to be defined at all, you can send the message

```
FileDirectory noDefault.
```

Notice that Smalltalk's volume prefixes are only one level deep: they may *not* designate a fully-qualified pathname of arbitrary depth. To access a volume nested several levels deep in the folder hierarchy, you can use the special prefix

```
?:
```

When Smalltalk encounters this prefix in a file name (or when the file name has no prefix and no default volume is currently defined), it displays a menu of volume names on the screen like that used by the `choose volume` command in a file-list browser (see Chapter 2). You can now proceed to navigate your way through the folder hierarchy to the desired volume in the same way as with the `choose volume` command. Just as in `choose volume`, the volume you designate, as well as all others you choose along the way, become known to the system and can thereafter be used as simple, one-level prefixes. In addition, removable 3.5-inch disk volumes become known to the system when they're inserted in a drive and remain known even if later ejected.

You can define *aliases* for existing volume names by sending the message `alias:to:` to class `FileDirectory`. For example, the statement

```
FileDirectory alias: 'MyDisk' to: 'Internal'.
```

defines `MyDisk:` to be a valid volume prefix referring to the same volume as `Internal`. Once defined, an alias is completely equivalent to the original volume prefix. However, aliasing is not transitive: that is, after

```
FileDirectory alias: 'Blood' to: 'Sweat'.
FileDirectory alias: 'Sweat' to: 'Tears'.
```

the prefix `Blood:` is *not* equivalent to `Tears:` and no longer equivalent to `Sweat:`. To cancel an alias definition, send a message of the form

```
FileDirectory unalias: 'MyDisk'.
```

The following standard prefixes are built into the system:

Prefix	**Meaning**
`Internal`	Internal 3.5-inch disk drive
`External`	External 3.5-inch disk drive
`Device3`	Hard disk
`MacDefault`	Volume or folder from which Smalltalk was started up

(Notice that the name `MacDefault` is *not* the same as the default volume mentioned earlier. `MacDefault` always refers to the volume containing the Smalltalk image file, and is not affected by the `FileDirectory default:` message.) The volumes these refer to are known as *fixed volumes* and have some special properties. You may alias other names to these volumes or alias these names to other volumes, but the property of being a fixed volume belongs to the volume itself, not the prefix.

Instances of class `HFSMacVolume` respond to most of the standard Macintosh volume operations. In particular, the messages `eject`, `flush`, `mount`, and `unmount` are defined. (You can also eject a disk by choosing `eject disk` from the system menu, then selecting the volume you want to eject from the pop-up menu that appears.) Ejecting merely places the volume off-line but doesn't unmount it, so files may still be open on an ejected disk. On shutdown, all volumes are flushed and all open files are closed and placed in a state such that further activity on them will fail. Files will be reopened when the system comes up (if still referenced), but no check will be made to see if it is really the same volume.

For compatibility with the abstract Smalltalk model, open and `close` are also defined in class `HFSMacVolume`, and are equivalent to the standard Macintosh operations `GetVolInfo` and `FlushVol`, respectively. Smalltalk tries to keep you from wreaking havoc on the Macintosh's file system, but is far from foolproof. For instance, it's possible to unmount the Macintosh startup volume from within Smalltalk, with predictably disastrous results.

Files

Macintosh files are represented by objects of class `HFSMacFileStream`. This is a subclass of `FileStream`, the standard Smalltalk class denoting a file. Instances of `HFSMacFileStream` respond to the standard Smalltalk protocol for streams and files, as well as to other messages specific to Macintosh files, such as `finderInfo`, `setFileInfo:`, and `isVisible`. For a full discussion of streams and files, see Chapter 12 of ***Smalltalk-80: The Language and Its Implementation,*** by Goldberg and Robson (listed in Appendix E).

To open a file, send a message to class `FileStream` of the form

```
FileStream fileNamed: 'Hello'
```

This will open an existing file of the given name if there is one, otherwise create a new one. You can include a volume prefix if you wish, or omit it to open a file on the default volume. To limit the operation to an already existing file, use

```
FileStream oldFileNamed: 'Hello'
```

This will open a notifier if no such file exists, asking permission to create a new one. Similarly, you can use

```
FileStream newFileNamed: 'Hello'
```

to insist on a new file. If a file already exists by this name, a notifier will be opened asking for permission to overwrite it. Any time you do overwrite an existing file, you must remember to send it the message `shorten` before closing it, otherwise part of the old data may remain after the end of the new contents.

By default, all files created by Smalltalk have file type `TEXT`. This allows them to be read by any text editor that can handle text-only files, such as MDS Edit, the MPW editor, MacWrite, or Microsoft Word. They are automatically given MacWrite's creator signature, so that they will run MacWrite when double-clicked in the Finder. You can give a file a different type and creator, if you wish, by sending it the message `setType:creator:`.

Chapter 6

Accessing the Macintosh Toolbox

An important feature in this version of Smalltalk-80 for the Macintosh is its programming interface to the Macintosh User Interface Toolbox and Operating System. Smalltalk programs can call any routine in the Macintosh ROM: that is, any routine that is accessed using a trap. (Routines that are not accessed through traps are not currently supported.) To support this capability, this version of Smalltalk includes mechanisms for calling the ROM routines themselves and for manipulating the Pascal-style records and other data objects that these routines use. Most of the support code for Toolbox access is contained within the Smalltalk class categories `ToolBox-Support`, `ToolBox-Structures`, and `ToolBox-Heap Access`. For detailed information on the ROM routines themselves, see Apple's comprehensive *Inside Macintosh* manual.

Pascal data structures

Pascal-style data objects for use with the Toolbox are allocated directly from the Macintosh application heap, rather than from Smalltalk's own internal heap. When Smalltalk is first started up, it expands the application heap to its maximum size, reserves a fixed amount (normally 36K) for Toolbox objects, and allocates the rest as one large, nonrelocatable block for its own use. Within this private heap, Smalltalk allocates all of its normal (Smalltalk-style) objects and does its own memory management and garbage collection, independent of the Toolbox Memory Manager. Pascal-style objects are allocated in the remaining portion of the application heap and are accessed indirectly from within Smalltalk via "proxy" objects in the Smalltalk heap itself.

The Smalltalk classes that support access to Pascal-style data structures are summarized in the following subclass hierarchy:

```
Object ()
    PascalRecord ('handle' 'pointer' 'bits')
       HeapObject ()
           HeapArray ('logicalSize')
              EightInts ()
              EightLongs ()
              Pattern ()
           HeapRecord ()
              BitMap ()
              Finfo ()
              GrafPort ()
               HeapRecordPointers ('pointerArray')
                  HParamBlockRec ()
              Picture ()
              Region ()
              St255 ()
```

PascalRecord, HeapObject, HeapArray, HeapRecord, and HeapRecordPointers are defined in category ToolBox-Heap Access. All the rest—the classes representing actual Toolbox data structures such as Pattern, GrafPort, and HParamBlockRec—are in ToolBox-Structures.

The root of this hierarchy of classes is PascalRecord, which has three fields named handle, pointer, and bits. At any given time, at most one of these fields should be non-nil, depending on the nature of the object represented:

- □ If handle ~= nil, it holds a SmallInteger or LargePositiveInteger that is a handle (the address of a master pointer) to a relocatable block in the application heap.
- □ If pointer ~= nil, it holds a SmallInteger or LargePositiveInteger that is the address of a nonrelocatable block in the application heap.
- □ If bits ~= nil, it holds a ByteArray that is the direct representation of the object's value.
- □ If handle = pointer = bits = nil, the PascalRecord refers to no object at all.

For objects that reside in the Macintosh heap (handle ~= nil or pointer ~= nil), you must explicitly send the object the message release when you're through with it; this in turn calls the Toolbox trap DisposHandle or DisposPtr to deallocate the object from the heap. release is *not* called for you during garbage collection, so the pointer or handle in an object is lost if the object is garbage-collected before you send release. The space the object occupies in the Macintosh heap becomes unrecoverable for the remainder of your Smalltalk session.

Defining record types

Class HeapObject defines standard protocol for its subclasses HeapRecord and HeapArray. All Smalltalk classes representing Pascal record types are subclasses of HeapRecord. To define a new record class, send a message to class HeapRecord of the form

```
HeapRecord recordSubclass: #NewClassName
    fields: 'field1 field2 ...'
    types: 'type1 type2 ...'
    category: 'ToolBox-Structures'
```

Where field1, field2, . . . are the names of the record's fields and type1, type2, . . . designate the corresponding data types. Each type specification is either a one-character code representing a primitive type or the name of another HeapObject subclass (for an embedded record or array). The following table shows the code characters for the primitive types; embedded subrecords are discussed in a separate section below.

Character	Pascal type	Smalltalk type
I	INTEGER	SmallInteger, LargePositiveInteger up to 16 bits
L	LONGINT	SmallInteger, LargePositiveInteger up to 32 bits
B	BOOLEAN	Boolean
P	Point	Point
R	Rect	Rectangle
S	Str255	String
D	Ptr	PascalRecord (D for "direct pointer")
H	Handle	PascalRecord

Methods will automatically be defined in the new subclass for accessing and changing all of the fields you name in your definition message. These methods use Smalltalk's built-in primitive 161, which takes two arguments: a byte offset within the record and a one-character string designating the type of value to be found there. For example, class GrafPort includes the following methods for accessing the portRect field of the GrafPort record:

```
portRect
    <primitive: 161 recordOffset: 16 type: 'R'>
    ↑self primitiveFailed

portRect: rect
    <primitive: 161 recordOffset: 16 type: 'R'>
    ↑self primitiveFailed
```

The recordOffset value of 16 tells the primitive that the desired field begins at an offset of 16 bytes from the start of the record; the type string 'R' tells it that the value contained in that field is a rectangle. (The primitive can distinguish between the two messages by the number of arguments it receives on the stack.) The primitive: 161 part in these definitions is optional: the methods could just read

```
portRect
    <recordOffset: 16 type: 'R'>
   ↑self primitiveFailed

portRect: rect
    <recordOffset: 16 type: 'R'>
   ↑self primitiveFailed
```

instead.

If you select a HeapRecord subclass in the browser's class names pane and hold down the Shift key while choosing the definition command, a template will appear in the code pane showing the names and types of the record's Pascal-style fields.

❖ *Note:* Never hold the Shift key down while filing out a class.

Subrecords

When a Pascal-style record contains embedded subrecords or other data structures (such as the BitMap record embedded in the portBits field of a GrafPort), the problem of accessing these fields is less straightforward than for the simple data types listed in the table above. Ideally, when you ask for the contents of such a field, you should get back a new object that refers to the same data as in the original record (wherever it resides), so that garbage collection works reasonably and changes to the subrecord are properly reflected in the parent record.

That's just what happens with objects that reside in the Macintosh heap. A new instance of class PascalRecord is created, with its pointer field set to point to the desired subrecord at the appropriate offset within the parent record. You can then use this object to access or change the contents of the subrecord just as if it were a separate object.

There are risks to this approach, however. If the parent record is subsequently deallocated or moved within the Macintosh heap, your pointer to the subrecord will become invalid. Furthermore, you can sabotage the Toolbox Memory Manager if you try to treat the subrecord as if it were an independently allocated object. If you send the subrecord object the message release, for instance, the Toolbox trap DisposPtr will be called with a value that points into the middle of a heap block instead of the beginning, and a system error will result.

In general, rather than keep around a subrecord object that points into the middle of the parent record, it's better to create a new instance of the subrecord and copy the original data to it. In fact, for records residing in the Smalltalk heap, that's what you get in the first place. Again, a new instance of `PascalRecord` is created, but with a *copy* of the data instead of a pointer into the middle of the original record. In this case, changes to the subrecord are *not* reflected in the parent record. If you want to set a field of a subrecord, you have to get the subrecord with the appropriate access method, set the field, then store the modified subrecord back into the parent record.

For example, suppose `pbRec` is an instance of class `HParamBlockRec`, representing a parameter block for use with the Toolbox File Manager. Field `ioFlFndrInfo` is a subrecord of the parameter block, of type `Finfo`, holding the file's Finder information. If you want to set the `fdCreator` field of the `Finfo` record, you can't do it with a straightforward message such as

```
pbRec ioFlFndrInfo fdCreator: 'MACA'.
```

because the object returned by `pbRec ioFlFndrInfo` is a *copy* of the `Finfo` subrecord, not a reference to the embedded subrecord itself. So to set the `fdCreator` field, you have to do it in a more roundabout way:

```
finderInfo ← pbRec ioFlFndrInfo.
finderInfo fdCreator: 'MACA'.
pbRec ioFlFndrInfo: finderInfo.
```

Currently, class `HParamBlockRec` is the only one that allocates its data on the Smalltalk heap. `HParamBlockRec` is a subclass of `HeapRecordPointers`, which is used to pass the addresses of buffers and other data objects that exist in the Smalltalk heap but are pointed to by fields of a Toolbox record structure. If Smalltalk's memory manager moves such an object within the Smalltalk heap, the pointer in the Toolbox structure will be left pointing to the wrong address. `HeapRecordPointers` avoids this problem by substituting the correct addresses of such objects at the time of the Toolbox call, within the low-level primitive that implements the call.

For example, suppose one of the fields of the Pascal record `fooRecord` is a pointer to a Smalltalk byte array named `fooBuffer`. You would use an object of class `HeapRecordPointers` with a pointer array whose first element is the offset of that field within `fooRecord`, and whose second element refers to the object `fooBuffer` itself. During the call in which `fooRecord` is passed to the Toolbox as a parameter, the primitive will calculate the current address of `fooBuffer` in the Smalltalk heap and store it at the specified offset in the record before passing the record to the Toolbox. Thus the Toolbox is guaranteed to get the correct address of `fooBuffer` at the time of the call.

Class `HeapRecordPointers` is tricky to use and skimpily documented, and you should avoid it if at all possible. If you absolutely must use it, be sure to study it carefully beforehand, and be forewarned.

Heap inspectors

A handy new feature in this release is class `HeapInspector`, which allows you to open a Smalltalk-type inspector window on any object that belongs to a subclass of `HeapRecord`. You can now examine and modify all the fields of a Pascal record in the same way you can for a standard Smalltalk object. Just send the message `inspect` to any object belonging to a `HeapRecord` subclass.

Heap inspectors use the list of field names and field types stored in each heap object's class variables to access the individual fields of the record and display their names. Class `HeapRecord` uses the same lists in its subclass definition method (`recordSubclass:fields:types:category:`), to compute the offsets of all fields and automatically generate methods for accessing and setting them.

Calling the Toolbox

In general, you issue calls to the Macintosh Toolbox by sending messages to the Smalltalk object `Mac`, the only instance of class `Macintosh` (defined in category `ToolBox-Support`). The first keyword of the message selector corresponds to the name of the Toolbox routine as given in *Inside Macintosh,* but beginning with a lowercase letter. For example, for the Toolbox routine defined in Pascal as

```
PROCEDURE FrameRect (r: Rect);
```

a Pascal call such as

```
FrameRect (myRect)
```

would correspond to the Smalltalk message

```
Mac frameRect: myRect.
```

If the Toolbox routine has more than one parameter, the second and subsequent parameters are converted into keywords in the Smalltalk message: for the Pascal routine

```
PROCEDURE FrameRoundRect (r: Rect; ovalWidth, ovalHeight: INTEGER);
```

the Pascal call

```
FrameRoundRect (myRect, w, h)
```

converts to the Smalltalk message

```
Mac frameRoundRect: myRect ovalWidth: w ovalHeight: h.
```

A Toolbox routine that takes no parameters at all corresponds to a unary message in Smalltalk: to call the Pascal routine

```
PROCEDURE HidePen;
```

you would use the Smalltalk message

```
Mac hidePen.
```

Primitive 160

Calls to the Toolbox are implemented via two interpreter primitives, numbers 160 and 162. The first, primitive 160, takes two parameters: an integer (usually expressed in hexadecimal form) representing the last three digits of the Toolbox trap word, and a coded string of characters describing the arguments to the Toolbox routine. For example, the Toolbox routine `FrameRect`, whose Pascal declaration was given earlier as

```
PROCEDURE FrameRect (r: Rect);
```

is defined in Smalltalk, in class `Macintosh`, as follows:

```
frameRect: r
    <primitive: 160 trapA: 16rCA1 type: '--R'>
    ↑self primitiveFailed
```

Here the `trapA` parameter gives the last three hexadecimal digits of the trap word; the first digit is always understood to be `A`, so the complete trap word, in Pascal notation, is `$ACA1`. The `type` parameter is coded according to conventions that we'll be discussing in a minute; in this case, it specifies that the routine has one argument of Pascal type `Rect` and returns no function result. The primitive will check the type of the argument passed on the Smalltalk stack and signal failure if it doesn't match. Otherwise it will convert the value to Pascal format, push it onto the Macintosh stack, and call the designated trap.

The `type` parameter to primitive 160 is a string containing a sequence of individual *type specifiers*. Each specifier consists of one of the eight code letters representing the primitive data types (`I`, `L`, `B`, `P`, `R`, `S`, `D`, or `H`), as shown in the table above under "Defining Data Types." This type letter may optionally be preceded by a `V`, denoting a `VAR` parameter. For parameters passed in registers rather than on the stack, the type letter is followed by a digit specifying the register. The Toolbox uses three registers for parameter passing, denoted by the following code digits:

Digit	Register
0	D0
1	A1
2	A0

Thus the type specifier VL0 would represent a VAR parameter of type LONGINT, passed in register D0. In the string passed as the type parameter to primitive 160, the first type specifier represents the function result returned by the Toolbox routine (or - for procedures that don't return a result); the second denotes the Smalltalk object to which the message is sent (or - if the message receiver is not a parameter of the Toolbox routine); and the rest of the string gives the types of the remaining parameters. Some more examples should help make all this clear:

Pascal

```
PROCEDURE GetPort (VAR gp: GrafPtr);
```

Smalltalk

```
getPort:  gp
    <primitive: 160
     trapA: 16rC74    "trap word = $AC74"
     type: '--VD'>    "no result, ignore receiver, VAR pointer"
   ↑self primitiveFailed
```

Pascal

```
PROCEDURE OpenPort (gp: GrafPtr);
```

Smalltalk

```
openPort:  gp
    <primitive: 160
     trapA: 16rC6F         "trap word = $AC6F"
     type: '--D'>          "no result, ignore receiver, pointer"
   ↑self primitiveFailed
```

Pascal

```
FUNCTION SectRect (srcRectA, srcRectB: Rect; VAR dstRect: Rect)
                      : BOOLEAN;
```

Smalltalk

```
sectRect:  srcRectA  srcRectB:  srcRectB  dstRect:  dstRect
    <primitive: 160
     trapA: 16rCAA          "trap word = $ACAA"
     type: 'B-RRVR'>        "Boolean result, ignore receiver,
                             rectangle, rectangle, VAR rectangle"
   ↑self primitiveFailed
```

Pascal

```
FUNCTION NewPtr (logicalSize: Size): Ptr;
```

Smalltalk

```
newPtr:  logicalSize
    <primitive: 160
     trapA: 16r01E          "trap word = $A01E"
     type: 'D2-L0'>         "pointer result in A0, ignore receiver,
                             long integer in D0"
   ↑self primitiveFailed
```

Pascal

```
PROCEDURE SetPt (VAR pt: Point; h,v: INTEGER);
```

Smalltalk

```
setPt: pt h: h v: v
    <primitive: 160
     trapA: 16rC80         "trap word = $AC80"
     type: '--VPII'>       "no result, ignore receiver,
                            VAR point, integer, integer"
   ↑self primitiveFailed
```

Instead of defining this last method in class `Macintosh`, it could instead be put in class `Point`. Instead of

```
Mac setPt: thePoint h: horiz v: vert
```

you would write

```
thePoint setH: horiz v: vert
```

In this case the message receiver itself is one of the arguments to be passed to the Toolbox routine, so the method definition in class `Point` would be

```
setH: h v: v
    <primitive: 160
     trapA: 16rC80         "trap word = $AC80"
     type: '-VPII'>        "no result, receiver is VAR point,
                            integer, integer"
   ↑self primitiveFailed
```

As with primitive 161, the `primitive: 160` in all these definitions may be omitted. For example, the last method could have been written as

```
setH: h v: v
    <trapA: 16rC80 type: '-VPII'>
   ↑self primitiveFailed
```

Primitive 162

If you browse class `Macintosh`, you'll find that most of the messages corresponding to Toolbox calls are not explicitly defined there. Instead, `Macintosh` redefines the message `doesNotUnderstand:` (which normally displays a notifier on the screen reading `Message not understood`) to intercept all unrecognized messages and pass them to another Smalltalk primitive, number 162. Instead of taking explicit parameters for the trap word and parameter types, this primitive looks up the undefined message selector in an internal table of Toolbox traps. (This search uses only the first eight characters of the selector and ignores upper- and lowercase differences.) If it finds an entry for this selector, it gets the appropriate trap number and parameter information from the table and completes the call; if not, it posts a notifier on the screen.

Calling a Toolbox routine with primitive 162 takes longer than with primitive 160, because of the table lookup. However, the explicit method definitions needed to use primitive 160 use up space within your Smalltalk image. For this reason, it's recommended that you use primitive 162 for most Toolbox calls, and 160 only for those that you use very frequently or for which speed is important, such as on-screen graphics. You can also use primitive 160 to define methods for new routines recently added to the Toolbox in the Macintosh Plus ROM. (In fact, because the Toolbox lookup table was not updated for this release, this is how all the new HFS file system calls were added to the system.)

Another use for primitive 160 is to fix bugs in the internal lookup table used by primitive 162. If an attempted Toolbox call causes a primitive failure or a system crash, there's probably something wrong with one of the parameters you passed to it. (Remember, for instance, that `VAR` parameters have to be preinitialized to something valid before being passed to the Toolbox.) However, it's also possible that the call you're trying to use is missing or encoded wrong in the lookup table; not all of the Toolbox calls were exhaustively tested before this release of the system was shipped. In this case, you can correct the problem by adding an explicit method for the trap using primitive 160.

For example, the table entry for the Toolbox routine `GetNextEvent` is known to be incorrect. This routine, defined in Pascal as

```
FUNCTION  GetNextEvent(eventMask:  INTEGER;
                       VAR theEvent: EventRecord)  : BOOLEAN;
```

is incorrectly encoded in the lookup table as `B-IVD`; it should actually be `B-ID`, since `theEvent` is declared as an event record, not a pointer to a record. (Under Pascal conventions, a pointer to the event record is passed on the stack, but the routine only changes the contents of the record, not the value of the pointer.) This error could be corrected (though it isn't in the actual release) by defining the following method in class `Macintosh`:

```
getNextEvent:   eventMask  theEvent:   theEvent
    <primitive: 160
     trapA: 16rD70          "trap word = $AD70"
     type: 'B-ID'>          "Boolean result, ignore receiver,
                              integer, pointer"
   ↑self primitiveFailed
```

Things to watch out for

Here are a few potential pitfalls to keep in mind when using the Toolbox from Smalltalk:

- □ You'll probably need access to various Toolbox globals and constants. Methods for some of these are defined in class `Macintosh` in the message category `globals/constants`; those that aren't defined there are available with the `StdPools` goodie, described in Appendix B. Also look at category `memory inspect` in class `Macintosh` to see how to access or set specific memory locations from within Smalltalk.
- □ Calls to Toolbox routines that return `VAR` parameters can sometimes lead to subtle problems when translated into Smalltalk. For example, the Pascal statement

  ```
  GlobalToLocal  (theEvent.where)
  ```

  corresponds to the Smalltalk expression

  ```
  Mac globalToLocal: theEvent where.
  ```

  In this case, the `where` field of event record `theEvent` will never get the value returned to it, since the message `theEvent where` returns a temporary object that receives the new value returned to it. The correct way to code this is

  ```
  tempPoint ← theEvent where.
  Mac globalToLocal: tempPoint.
  theEvent where: tempPoint.
  ```
- □ When using data in the Smalltalk/Toolbox environment, you must not pass the Toolbox the address (pointer or handle) of any data stored in the Smalltalk heap if the Toolbox is going to remember the address after returning from that one call. Although the location of a Smalltalk `ByteArray` object can't change during the time a single Toolbox call is running, it is possible for it to change between calls. So byte arrays should be used only for data whose address is passed to the Toolbox again every time the Toolbox uses it. This ensures that the correct address will be used each time.
- □ In Smalltalk you can't write new values into a procedure's parameters, as you can in Pascal. Instead you must create a local temporary variable, along with the code to copy the parameter's value into that temporary. Within the method, you can then assign the new value to the temporary.

- ☐ Toolbox routine names that are defined in Primitive 162's lookup table may not be known to the spelling checker. The first time you try to use such a name as a message selector, the spelling checker will reject it and offer you a list of alternative spellings. When this happens, just accept the spelling as is; the spelling checker will then remember it and accept it without protest the next time you use it.

Appendix A

System Workspace Summary

Here is a summary of useful Smalltalk expressions that are available in the System Workspace window:

Changes and Files

`Smalltalk noChanges.`	Resets the change set to empty.
`Smalltalk condenseChanges.`	Condenses the changes file, removing multiple copies of method definitions, methods no longer accessible from the current image, and other unnecessary text. May take a long time, and requires sufficient disk space for two copies of the changes file. It's advisable to make backup copies of your image and changes before condensing and take a snapshot immediately afterward.
`DisplayScreen removeFromChanges.`	Removes all changes in a given class from the change set.
`Smalltalk changes asSortedCollection`	Returns a sorted list of the current contents of the change set.
`Smalltalk browseChangedMessages.`	Opens a message browser on all methods in the change set.

`(FileStream fileNamed: 'Changes.st') fileOutChanges.`
Files out all changes in the change set to a given file.

`(FileStream fileNamed: 'PenChanges.st') fileOutChangesFor: Pen.`
Files out all changes in the change set for a given class.

`(FileStream oldFileNamed: 'BrowseFeatures3.st') fileIn.`
Files in changes from a given file.

`(FileStream fileNamed: 'Hello') edit.`
Opens a file editing window on a given file.

`FileDirectory filesMatching: '*.st'`
Returns a list of all filenames matching a given pattern. (Actually, it's more convenient to open a file-list window and type the pattern into the top pane, followed by Return.)

`ChangeList browseFile: 'changes.st'.`
Opens a change-list browser on a given file.

`ChangeList browseRecentLog.`
Opens a change-list browser on all changes in the system changes file since the last snapshot.

Inquiry

`InputState browseAllAccessesTo: 'deltaTime'.`
Opens a browser on all references in a class to a given instance variable. (Like `inst var refs` in the system browser.)

`Smalltalk browseAllCallsOn: #r.`
Opens a browser on all calls to a given message selector. (Like `senders` in the system browser.)

`Smalltalk browseAllImplementorsOf: #remove:.`
Opens a browser on all method definitions for a given message selector. (Like `implementors` in the system browser.)

`Smalltalk browseAllCallsOn: (Smalltalk associationAt: #ShutDownList).`
Opens a browser on all references to a given global variable.

`Smalltalk browseAllCallsOn: (Cursor classPool associationAt: #ReadCursor).`	Opens a browser on all references in a class to a given class variable.
`Smalltalk browseAllCallsOn: (Undeclared associationAt: #Disk).`	Opens a browser on all references to a given undeclared variable.
`Smalltalk browseAllMethodsInCategory: #examples.`	Opens a browser on all methods in a given message category. (May cover more than one class.)
`(Smalltalk collectPointersTo: StrikeFont someInstance) inspect.`	Opens an inspector on all objects that point to an instance of a given class.
`Smalltalk garbageCollect.`	Forces a full garbage collection of the Smalltalk heap.
`FileStream instanceCount`	Returns the number of instances of a given class currently in existence.
`FormView allInstances inspect.`	Opens an inspector on all instances of a given class.

HouseCleaning

`Undeclared ← Dictionary new.`	Resets the dictionary of undeclared variables to empty. Note that you will no longer be able to find the methods in which they occur.
`Undeclared inspect.`	Opens an inspector on all undeclared variables.
`Undeclared associationsDo: [:assn | Smalltalk browseAllCallsOn: assn].`	Browses all references to undeclared variables.
`(Object classPool at: #DependentsFields) keys`	Returns a set of all objects with dependents.

```
(Object classPool at: #DependentsFields) keysDo:
        [:each | (each isKindOf: DisplayText)
                          ifTrue: [each release]].
```

Releases all objects that satisfy a given condition.

```
Transcript clear.
```

Resets the transcript window to empty.

```
Smalltalk forgetDoIts.
```

Removes methods generated by `doIt` and `printIt`.

```
Symbol rehash.
```

Purges any symbols that are no longer in use.

```
Smalltalk removeKey: #GlobalName.
```

Removes a symbol from the global dictionary. You should check first that there are no outstanding references to it.

```
Smalltalk declare: #GlobalName from: Undeclared.
```

Declares a symbol as a new global variable. If it was formerly in the `Undeclared` dictionary, all references to it will now refer to the new global variable. (This is how forward references are handled in file-ins.)

Globals

```
Names in Smalltalk other than Classes and Pools:
        Display -- a DisplayScreen
        Processor -- a ProcessorScheduler
        ScheduledControllers -- a ControlManager
        Sensor -- an InputSensor
        Transcript -- a TextCollector
        ScurceFiles -- an Array of FileStreams
        SystemOrganization -- a SystemOrganizer
        StartUpList -- an OrderedCollection
        ShutDownList -- an OrderedCollection
```

```
Variable Pools (Dictionaries)
        Smalltalk
        Undeclared
        FilePool
        OSIntfPool
        BitMaskPool
        TextConstants
```

System Files

```
SourceFiles ← Array
        with: (FileStream oldFileNamed:
                                Smalltalk sourcesName) readOnly
        with: (FileStream oldFileNamed:
                                Smalltalk changesName).
```
Establishes the sources and changes files.

`(SourceFiles at: 1) close.`	Closes the sources file.
`(SourceFiles at: 2) close.`	Closes the changes file.
`SourceFiles ← Array new: 2.`	Disables the sources and changes files. (Sources can still be retrieved with the goodie `RetrieveSources`.)

Measurements

`Smalltalk spaceLeft`	Returns the amount of free heap space currently available.
`Symbol instanceCount`	Returns the number of instances of a given class currently in existence.

```
Time millisecondsToRun: [Smalltalk allCallsOn: #asOop]
```
Returns the time needed to execute a given block.

```
MessageTally spyOn: [Smalltalk allCallsOn: #asOop].
```
Presents a time-use profile for execution of a given block.

Crash recovery

`Smalltalk recover: 5000.`	Opens a window on the last 5000 characters in the changes file.

Appendix B

Smalltalk Goodies

Included as samples with this release are several additional source files, known as *goodies,* which add interesting or useful capabilities to your Smalltalk system. The goodies are divided into two categories, demos and utilities, and are found in the folders `Goodies-Demos`, on *Smalltalk Disk 2,* and `Goodies-Utilities`, on *Smalltalk Disk 7.*

To read a goodie named `Gumdrop` into your system, execute the statement

```
(FileStream oldFileNamed: 'Gumdrop.st') fileIn.
```

or select `Gumdrop.st` in a file list window and choose `fileIn` from the list pane menu. The filenames are not case-sensitive, but spaces are significant. The `.st` extension is only a convention, to make it easy to browse all Smalltalk files using the pattern `*.st` in the top pane of a file-list window.

Demos

The following goodies are found in the `Goodies-Demos` folder on *Smalltalk Disk 2:*

Fractal

Produces three-dimensional surfaces based on fractal geometry. For an example, execute

```
Fractal example,
```

Toothpaste

Allows you to draw shaded worm-like curves on the screen, using a "brush" that looks like a highlighted sphere. Execute

```
Form toothpaste: 30.
```

and then paint with the cursor. To stop, hold down the Option key and click the mouse.

Web

Another drawing program, which lags behind the current mouse position and then draws lines between the lagging cursor and the current cursor. Execute

```
QDPen new web.
```

and then draw with the mouse. Click the mouse to erase the screen, Option-click to stop.

Music

A music synthesizer that drives the Macintosh four-tone sound generator. See the MacWrite file `Music Docs` in the `Goodies-Demos` folder on *Smalltalk Disk 2* for more information.

Utilities

The following goodies are found in the `Goodies-Utilities` folder on *Smalltalk Disk 7:*

Printing

Defines a new class `MacPrintStream` to support hard-copy printing on an Apple Imagewriter™ printer connected to the printer port. Adds a menu command for printing in most windows. The method `examples` in `MacPrintStream class` gives general instructions for use. Printing from the window menu (in a window's title tab) produces a bit image of the window; printing from a text pane produces text, with an attempt to support the unusual Smalltalk characters. Also included is a method for rotating bit images by 90 degrees, for use if you want to implement landscape printing.

FFT

Defines a new class `FFT` for performing a one-dimensional fast Fourier transform on data held in Smalltalk arrays. The example method `test` illustrates its use. This code computes a complex Fourier transform; a corresponding power spectrum can be derived by summing the squares of corresponding real and imaginary components.

Macintosh-QuickDraw

A full set of QuickDraw call definitions using primitive 160 instead of 162 (see Chapter 6). This makes all the QuickDraw messages visible and easier to browse, at some cost in memory space.

MacPaint

Defines a method `macPaintOn:` in class `Form` to create a graphics file readable by MacPaint™. The comment at the end of the method gives a sample invocation. This method is already included in the released image and source, but its example comment is in error; this goodie corrects the comment.

RS232

Defines methods to support input and output via the two Macintosh serial ports. One of these includes a large comment explaining how to set baud rate and so forth; another is a sample application that downloads text (for example, from a laptop computer) to a Macintosh file.

RetrieveSources

Allows access to the full system source code, even without a hard disk. If you have browsed to a given method (decompiled) and wish to see the full source code (as you would with a hard disk), choose `retrieve` from the code pane menu. You will be asked to insert one of the disks containing the divided sources file, and then the full code will appear. This technique will fail if you happen to choose one of the three methods that straddle breaks between the file divisions.

DropChanges

For the fearless programmer whose changes file has grown too large (even after condensing), but who still wants to continue making changes. Simply deleting the changes file would prevent the further recording of changes. Instead, by executing `Smalltalk dropChanges`, you can discard the current contents of the changes file (save them first, if you care), but the file will continue to record further development.

StdPools

Creates pools `ToolIntfPool`, `PackIntfPool`, and `QuickdrawPool`, and adds to the existing `OSIntfPool`, so that all four match the latest Pascal interface files released with the Macintosh Programmer's Workshop (version B2). These pools add approximately 10K to the size of your Smalltalk image.

VersionIIMenus

This goodie makes the menus in version 0.3 more compatible with the menus in Xerox Smalltalk, Version II. It is especially useful if you are using any of the introductory books.

Appendix C

Memory Management Techniques

The interpreter furnished with this release uses garbage collection rather than reference counts to reclaim unused storage. For this reason, it is not necessary to break circular structures in this implementation, though such "release" code still exists in many parts of the system. You'll notice occasional pauses when garbage collection takes place. Still, the system's performance is considerably better with garbage collection than with reference counting. A side effect of incremental garbage collection is that `allInstances` and related enumeration messages will sometimes enumerate objects that are no longer truly accessible. If you want to be sure of accurate results, execute

```
Smalltalk garbageCollect.
```

in the System Workspace immediately before such enumeration.

Even with garbage collection, it is still possible to run out of memory. When space starts getting low (below about 1000 objects and 8000 words of data), Smalltalk attempts to gain some more by turning off fast window display. If there is enough space to do so, it will open a notifier window at this point to warn you that you're running out of space. It is generally advisable to close the notifier rather than proceed, since soft error recovery may not be possible. (If you get more than one such notifier, keep on closing them until you get no more.)

You can find out how much memory you have left with the expression

```
Smalltalk spaceLeft
```

in your System Workspace window. This returns the number of free objects and the number of free words of data. (The amount of space reported will generally be greater than at the time Smalltalk detected it was running low, because some extra space will have been gained by turning off fast windows.) For more specific, detailed information, use

```
Smalltalk printSpaceAnalysis.
```

This writes out a disk file names `STspace.text`, listing the approximate number of objects and words used by each class in the system.

The activation stack displayed in the low-space notifier will show whether the problem was caused by endless recursion. If not, you should consider how else you might have unintentionally consumed a lot of memory and what steps you can take to free some. The problem may have been caused by having too many windows open on the screen, or something in your code may be using a lot of space. If you really need to make more space available in the system, here are some things you can try:

□ Execute

  ```
  Smalltalk deleteClasses.
  ```

  which will remove some of the less important classes and other objects from the system. (You may want to look at this method first, in class `SystemDictionary`, to be sure you want to remove all these classes. Also note that the current release doesn't include class `SystemTracer`, so remove the reference to this class from the `deleteClasses` method before executing it.)

□ Remove the debugger's `step` and `send` commands, and any other code that uses the simulation in class `ContextPart`. If you can't live without `step` and `send`, then implement breakpoints (a useful thing to do anyway) and use these to restore the operation of `step` and `send`.

□ Make `MessageSet`, and maybe even `Debugger`, a subclass of `Browser` so that they can share code for printing, file-out, `senders`, and `messages`.

□ If you have a hard disk, remove class `Decompiler`. One minor problem is that the decompiler is currently used for viewing the code of `doIt` methods from the debugger. If you can't live without this, figure out how to log the text of `doIt` methods in the changes file and give them a proper source-code pointer.

- ☐ After removing significant parts of the system in this way, you can execute the statement

  ```
  Smalltalk removeUnSentMessages.
  ```

  This locates and removes any messages that are implemented but never sent from anywhere in the system. It's useful to run this method four or five times in a row, because each method removed may render other methods inaccessible. The method takes significant time (from 5 to 25 minutes) to run.

- ☐ Finally, you should execute

  ```
  Symbol rehash.
  ```

  which will reclaim all symbols that are no longer in use as a result of the removal of methods.

Leaving more space for the Toolbox

As mentioned in Chapter 6, the Smalltalk interpreter allocates most of the application heap as a single nonrelocatable block, which it uses for its own Smalltalk heap. The amount of space withheld from this block for Toolbox operations is determined by a resource of type `STOP` (for "Smalltalk options") in the interpreter's resource file, and is initially set to 36K. If you need more heap space for Toolbox objects and can spare it from your Smalltalk image, you can use the ResEdit resource editor to change this value.

The `STOP` resource includes four values of 4 bytes each, only the first two of which are relevant and alterable. The first is the amount of heap space set aside for Toolbox operations, normally 36K; the second is the amount of stack space, normally 12K. Note that you're taking a chance if you change these values. If your Smalltalk image is already large and you reserve more space for the Toolbox or stack, the interpreter may not have enought free space to run the image the next time you try to launch it.

Appendix D

Known Bugs and Limitations

The following is a list of known bugs and limitations in release 0.3 of Smalltalk-80 for the Macintosh. Please report any other problems you may encounter, using the bug report form at the end of this book.

- The (collapsed) system browser on the initial startup screen of this release is not correctly updated. Selecting class `PascalRecord` in the class names pane will cause an error, since this class has been moved to a different category. To correct the problem, choose `update` in the browser's class categories pane.
- Method `deleteClasses` in class `SystemDictionary` still refers to class `SystemTracer`, even though this class has been removed from the system. The reference to `SystemTracer` should be deleted from the first statement of this method.
- Level 0 images from older versions (release 0.2) of Smalltalk-80 for the Macintosh won't run with the new interpreter. Level 1 images will run, but you should keep the 0.2 interpreter around to run with level 0 images. You will also need to keep around two versions of the `Smalltalk-80.sources` file, since the 0.3 sources are new. To make sure both versions use the right sources, put the old sources in the root directory of your hard disk and the new ones in a folder along with your 0.3 interpreter and image.
- If you move your Smalltalk image to a system with a larger display, existing windows cannot be framed larger than the old screen. If you can recreate the window's contents easily, close and reopen it.
- For historical reasons, Smalltalk uses a few bytes of memory preceding its display object, the screen, for private purposes. But on the Macintosh, memory just below the screen may be used by other programs (MacsBug, the RAM cache, some file servers), and some of the new large screens have no memory there at all. To work around this problem, Smalltalk-80 for the Macintosh reserves the top scan line of the screen for its private use, and fools QuickDraw into thinking that the

screen is one pixel shorter than its true height, starting at the second scan line instead of the first. You cannot access the top line, nor can you change the few bits of it that are left white.

- □ The system transcript window is not always updated correctly after the message `Transcript clear`. Reframing the transcript window corrects it.
- □ The `choose volume` command in a file-list window sometimes doesn't correctly restore the area underneath the volume list.
- □ The `find window` command on the system menu has display problems if you select a window that's entirely or almost entirely off the screen. Once you choose the window from the `find window` menu and position it on the screen, you can correct it by reframing.
- □ The form editor is buggy.
- □ Filename patterns containing an asterisk (*) should be used only in the top pane of a file-list window. Don't use such patterns when specifying a filename from within a method.
- □ The `explain` command in the browser's code pane sometimes displays two statements for you to execute, but omits the period between them. Insert a period before attempting to execute both statements with `doIt`.
- □ If Smalltalk crashes and leaves the changes file (or some other file) open, you may get an error when you try to restart Smalltalk, since the file can't be opened a second time for write access. This doesn't normally happen; but if it does, restart the entire system with the Finder's `Shut Down` command before restarting Smalltalk.
- □ Holding down the mouse button during a snapshot may cause the system to return from the snapshot with the mouse out of sync.
- □ The system may occasionally hang during a large file-in. If this happens to you, restart Smalltalk and start the file-in over again. If you're filing in many files at once, try filing each one in separately and doing a snapshot after each.
- □ If you start up Smalltalk under older versions of the Finder and system file, it may not open the sources and changes files properly. One indication of this problem would be decompiled instead of fully commented source code in the browser. If you encounter this problem, first make sure the file `Smalltalk-80.sources` is either in the root directory of your hard disk or in the same folder as your image file. If it is, try starting Smalltalk and immediately quitting and saving changes. If the problem persists, check the version numbers of the Finder and System file and make sure you're using at least versions 5.3 and 3.2 respectively. If not, use the Installer utility to install the newer versions from the system folder on *Smalltalk Disk 1,* then try another quit-with-save, followed by a restart. If your system sources are still inaccessible, please report the problem on the bug report form at the back of this book.

- There are some known bugs (and there may be others not yet discovered) in the internal table used by primitive 162 to call Macintosh Toolbox routines (see Chapter 6).

Appendix E

Bibliography of Smalltalk Literature

Following is a bibliography of available literature on Smalltalk and object-oriented programming:

Introductory

Goldberg, Adele. ***Smalltalk-80: The Interactive Programming Environment.*** Reading, Mass.: Addison-Wesley, 1984. Describes the user interface of Smalltalk, such as browsers and debugging tools. Sometimes referred to as the "orange book." This book and the one below by Goldberg and Robson together offer a comprehensive description of Smalltalk-80.

Kaehler, Ted and Patterson, Dave. ***A Taste of Smalltalk.*** New York, N.Y.: W. W. Norton & Co., 1986. A friendly introduction to the subject, including a detailed treatment of an actual programming problem. Points out differences between Smalltalk-80 Version I (Macintosh) and Version II (Xerox).

❖ *Note:* If you are using either of these introductory books, you may want to use the goodie `VersionIIMenus` described in Appendix B; it makes the menus more compatible with Xerox Version II.

Advanced

Goldberg, Adele and Robson, David. *Smalltalk-80: The Language and Its Implementation.* Reading, Mass.: Addison-Wesley, 1983. Describes the Smalltalk language and system classes, with many examples. Sometimes referred to as the "blue book." A companion volume to the book by Goldberg listed above.

Krasner, Glenn, ed. *Smalltalk-80: Bits of History, Words of Advice.* Reading, Mass.: Addison-Wesley, 1983. A collection of papers describing several implementations of Smalltalk on different computers. For implementors of Smalltalk systems, not users. Sometimes referred to as the "green book."

Doyle, Ken; Haynes, Barry; Lentczner, Mark; Rosenstein, Larry. "An Object-Oriented Approach to Macintosh Application Development." Paper presented at the 3rd Working Session on Object-Oriented Languages, Paris, France, January 8-10, 1986. Describes the advantages of object-oriented programming for application development on the Macintosh. Gives the relationship betweeen Smalltalk, Object Pascal, and MacApp.

General interest

BYTE Magazine, August 1981. This entire issue is devoted to Smalltalk.

BYTE Magazine, August 1986. This entire issue is devoted to object-oriented languages, and includes a condensed version of *A Taste of Smalltalk,* by Kaehler and Patterson (see above).

Schmucker, Kurt J. *Object-Oriented Programming for the Macintosh.* Hasbrouck Heights, N.J.: Hayden Book Company, 1986. Contains an entire chapter devoted to Smalltalk-80 for the Macintosh.

"Smalltalk Comes to the Micro." *Computer Language* Magazine, August 1985.

Smalltalk-80™ for the Macintosh\
Institutional License Agreement

This Agreement covers an institutional license for Smalltalk-80. Please check what License you are applying for, fill in the number of machines for your institution, and sign this Agreement. This Agreement with the appropriate fee are to be returned to:

Apple Computer, Inc.
20525 Mariani Avenue, M/S 28B
Cupertino, CA 95014
Attn: Software Licensing Department

License	Fee	Check One
Institutional license to make copies for a limited number of machines	$150.00	________
Special rate for educational institutions	$ 50.00	________

Institutional license
You have ordered an institutional license to use Smalltalk-80 for the Macintosh, version 0.2 and/or version 0.3 (hereinafter "system") on _______ machines within your institution.

Apple grants to you the right to make that many copies of the system for distribution and use within your institution. You agree that the system shall only be distributed and used within your institution, and that the system will be used only in conjunction with the Macintosh Smalltalk interpreter, and running on Apple computer equipment.

You have read and agreed to the conditions stated above.

Name:_______________________________________
(please print)

Signed: ____________________________Date:__________

Co./University: _________________________________

Address:_______________________________________

City/State/Zip:____________________________________

9/26/86

Smalltalk-80™ for the Macintosh Institutional License Agreement

This Agreement covers an institutional license for Smalltalk-80. Please check which type of License you are applying for, fill in the number of machines for your institution, and sign this Agreement. Return this Agreement with the appropriate fee to:

Apple Computer, Inc.
20525 Mariani Avenue, M/S 28B
Cupertino, CA 95014
Attn: Software Licensing Department

License	Fee	Check One
Institutional license to make copies for a limited number of machines	$150.00	______
Special rate for educational institutions	$ 50.00	______

Institutional license

You have ordered an institutional license to use Smalltalk-80 for the Macintosh, version 0.2 and/or version 0.3 (hereinafter "system") on ______ machines within your institution.

Apple grants to you the right to make that many copies of the system for distribution and use within your institution. You agree that the system shall only be distributed and used within your institution, and that the system will be used only in conjunction with the Macintosh Smalltalk interpreter, and running on Apple computer equipment.

You have read and agreed to the conditions stated above.

Name:____
(please print)

Signed: ______________________________Date:______________

Co./University: ______________________________________

Address:____

City/State/Zip:_______________________________________

10/12/86

Smalltalk-80™ for the Macintosh™
Bug Report Form

Smalltalk-80 is a trademark of Xerox Corporation. Macintosh is a trademark of Apple Computer, Inc.

Return to: The Smalltalk Group (MS 22-Y)
Apple Computer, Inc.
20525 Mariani Ave.
Cupertino, CA 95014

AppleLink: Smalltalk Group care of SARDI1
UNIX: ...!apple!smalltalk

Reported by: Name ____________ Company ____________

Address ____________ State______Zip_____

Phone ____________

Date ____________

System: Macintosh Plus_____ Macintosh XL _____ Other________ Memory ________

Disk Drives: Internal (400K/800K)________ External (400K/800K)________

Apple Hard Disk 20 ________ Apple Hard Disk 20SC ________

Other (please specify) ____________

Other Peripherals:____________

Describe any hardware upgrades/modifications:____________

ROM version:__________ System file Version: __________ Finder Version: __________

Smalltalk Version Number:__________

What other software (debugger, RAM disk, etc.), if any, was running? ____________

____

Describe the problem (specific steps/examples): ____________

____

____

____

YOU SHOULD CAREFULLY READ THE FOLLOWING TERMS AND CONDITIONS BEFORE USING THIS SOFTWARE. ANY DOWNLOADING, REPRODUCTION, COPYING OR OTHER USE OF THE SOFTWARE WILL CONSTITUTE ACCEPTANCE OF THESE TERMS AND CONDITIONS.

SINGLE-COMPUTER END USER SOFTWARE LICENSE AGREEMENT

APPLE COMPUTER, INC. ("Apple") provides this software and licenses its use. You assume responsibility for the selection of the software to achieve your intended results, and for the installation and use of, and results obtained from, the software.

LICENSE

Pursuant to this license you may:

1. Use the software only on a single Apple computer. You must obtain a supplementary license from Apple before using the software in connection with systems and multiple central processing units, computer networks or emulations on mainframe or minicomputers.
2. Download the software only on media that is compatible with Apple manufactured computers.
3. Copy the software into any machine readable form for backup purposes in support of your use of the software on the single Apple computer.
4. Transfer the software and license to another party with a copy of this Agreement provided the other party reads and agrees to accept the terms and conditions of this Agreement. If you transfer the software, you must at the same time either transfer all copies, whether in printed or machine-readable form, to the same party or destroy any copies not transferred. Apple grants a license to such other party under this Agreement and the other party will accept such license by its initial use of the software. If you transfer possession of any copy of the software, in whole or in part, to another party, your license is automatically terminated.

This software is protected by United States copyright law. You must reproduce the Apple copyright notice on any copy of the software.

THIS SOFTWARE MAY BE ELECTRONICALLY DISTRIBUTED ONLY BY AUTHORIZED ELECTRONIC DISTRIBUTORS. IT MAY BE DOWNLOADED ONLY FOR PERSONAL OR NON-COMMERCIAL USES ON APPLE COMPUTERS AND MAY NOT BE REDISTRIBUTED OR USED FOR COMMERCIAL PURPOSES WITHOUT AN EXPRESS SOFTWARE DISTRIBUTION LICENSE FROM APPLE. These licenses are available from Apple's Software Licensing Department.

YOU MAY NOT MODIFY, REVERSE COMPILE, DISASSEMBLE, NETWORK, RENT, LEASE, LOAN OR DISTRIBUTE THE SOFTWARE, OR ANY COPY, IN WHOLE OR IN PART. YOU UNDERSTAND THAT UNAUTHORIZED REPRODUCTION OF COPIES OF THE SOFTWARE OR UNAUTHORIZED TRANSFER OF ANY COPY OF THE SOFTWARE MAY SUBJECT YOU TO A LAWSUIT FOR DAMAGES, INJUNCTIVE RELIEF, AND ATTORNEY'S FEES.

Apple reserves all rights not expressly granted to you.

Export law assurances

You agree and certify that neither the software and documentation nor any direct product thereof (1) is intended to be used for nuclear proliferation or any other purpose prohibited by the United States Export Administration Act of 1979, as amended (the "Act") and the regulations promulgated thereunder, and (2) is being or will be downloaded, shipped, transferred or reexported, directly or indirectly, into any country prohibited by the Act and the regulations promulgated thereunder.

Government End Users

If you are acquiring the software on behalf of any unit or agency of the United States government, you agree that: (a) the software is "Commercial Computer Software" as that term is defined in Paragraph 27.401 of the DoD Supplement to the Federal Acquisition Regulations (the "Supplement") or is within the equivalent classification of any other federal agencies' regulations; (b) the software was developed at private expense, and no part of it was developed with goverment funds; (c) the government's use of the software is subject to "Restricted Rights" as that term is defined in clause 52.227-7013 (b)(3)(ii) of the Supplement or in the equivalent clause of any other federal agencies' regulations; (d) the software is a "trade secret" of Apple for all purposes of the Freedom of Information Act; and (e) each copy of the software will contain the following Restricted Rights Legend:

"Restricted Rights Legend"

Use, duplication or disclosure is subject to restrictions as set forth in subdivision (b)(3)(ii) of the Rights in Technical Data and Computer Software clause at FAR 52.227-7013. Manufacturer: Apple Computer, Inc. 20525 Mariani Avenue, Cupertino, Calfornia 95014.

You agree to indemnify Apple for any liability, loss, costs and expense (including court costs and reasonable attorneys' fees) arising out of any breach of the provisions of this Agreement relating to use by the government.

Term

The license is effective until terminated. You may terminate it at any time by destroying the software together with all copies. The license will also terminate upon conditions set forth elsewhere in this Agreement or if you fail to comply with any of the terms or conditions of this Agreement. You agree upon such termination to destroy all copies of the software.

Disclaimer of Warranty

The software is provided "as is" without warranty of any kind, either express or implied, with respect to its merchantability or its fitness for any particular purpose. The entire risk as to the quality and performance of the software is with you. Should the software prove defective, you (and not Apple or an Apple authorized representative) assume the entire cost of all necessary servicing, repair or correction.

Apple does not warrant that the functions contained in the software will meet your requirements or that the operation of the software will be uninterrupted or error free or that defects in the software will be corrected.

Some states do not allow the exclusion of implied warranties, so the above exclusion may not apply to you. This warranty gives you specific legal rights and you may also have other rights which vary from state to state.

Limitation of Remedies

In no event will Apple be liable to you for any lost profits, lost savings or other incidental, special or consequential damages arising out of the use of or inability to use any software even if Apple or an authorized Apple representative has been advised of the possibility of such damages, or for any claim by any other party.

Some states do not allow the limitation or exclusion of liability for incidental or consequential damages so the above limitation or exclusion may not apply to you.

Apple's liability to you for actual damages for any cause whatsoever, and regardless of the form of the action, will be limited to the greater of $500 or the money paid for the software that caused the damages or that is the subject matter of, or is directly related to, the cause of action.

General

This Agreement, if any attempt to network, rent, lease, or sublicense the software, or, except as expressly provided in this Agreement, to transfer any of the rights, duties or obligations under this Agreement, becomes void.

The Agreement will be construed under the laws of the state of California, except for that body of laws dealing with conflict of laws. If any provision of this Agreement shall be held by a court of competent jurisdiction to be contrary to law, that provision will be enforced to the maximum extent permissible, and the remaining provisions of this Agreement shall remain in full force and effect.