Frontier User Manual

UserLand Frontier Manual

Contents

HTML reformatting by Steven Noreyko. January 1996

Userland Frontier User Manual

Table of Contents

HTML reformatting by Steven Noreyko January 1996

Userland Software

Software: Dave Winer, Doug Baron
Project Team: Alice Lankester, Judi Lowenstein
Documentation: Dan Shafer

UserLand Software, Inc.
490 California Avenue
Palo Alto, CA 94306
U.S.A.
Telephone: 415/325-5700
Facsimile: 415/325-9829
AppleLink: USERLAND.DTS

First Edition: January, 1992

We would particularly like to thank the following people:

Randy Battat, Tracy Beiers, Eagle Berns, Tony Bove, Don Brown, Kevin Calhoun, Ben Calica, Dave Carlick, Datapak Software, Fred Davis, Pam Deziel, Chris Espinoza, Rich Gartland, Jean-Louis Gassee, Dave Jacobs, Christoph Jaggi, Bob LeVitus, Steve Levy, Rich McEachern, Steve Michel, Mike O'Connor, Michael Odawa, Tom Petaccia, Leonard Rosenthal, Jack Russo, Art Schumer, Francis Stanbach, Reede Stockton, Pete Stoddard, Michael Swaine, David Szetela, Terry Teague, Scott Trotter, Eve Winer, Leon Winer, Peter Winer, Mark Womack.

And to our worldwide family of beta sites, too numerous to mention here. Thanks!

© Userland Software, Inc. 1991, 1992. All rights reserved. No part of this publication may be reproduced in whole or in part, in any form without the express written permission of Userland Software, Inc.

UserLand Frontier, Frontier SDK, and UserTalk are trademarks of UserLand Software, Inc.

Contents Page | Preface

HTML reformatting by Steven Noreyko January 1996

Preface

Welcome to UserLand Frontier!

by Dave Winer
President, UserLand Software, Inc.

Frontier is a Scripting System

UserLand Frontier is the first scripting software that allows you to control the Macintosh operating system and System 7-compatible applications.

System-level scripting is an important component of most modern computer operating systems, but it has never been offered to users of graphic systems such as Apple's Macintosh.

Unix was the first widely used operating system to include scripting. When personal computers arrived, their operating systems also had integrated scripting. The Apple II implemented scripting with AppleSoft BASIC as its standard operating software. CP/M and MS-DOS include batch languages modeled after early minicomputer operating systems.

Then came the first commercial graphic personal computer, Apple's Macintosh. It quickly became famous for its ease of use. Instead of typing cryptic command-line instructions, the user would simply zoom out a folder containing a file and drag the file to the Trash icon. This approach resulted in significant gains in productivity. By getting the technology out of the user's way, these systems enabled people who were not technically inclined, to get something out of their computers.

The gains were great, but so was the loss. There was no way to teach a Macintosh to automatically do repetitive and time-consuming tasks. In this way, the Macintosh represented a step backward for sophisticated users.

This gap in the functionality of graphic operating systems is finally being filled by UserLand Frontier. With Frontier, power users can write scripts that automate and customize the system for themselves and for other users. Scripts can backup, rename, copy and delete files; and launch and control applications.

It may come as a surprise to you that underneath all the friendly icons, your Macintosh has a real operating system, waiting for a scripting product like Frontier to tap its power.

As you review the Frontier documentation you'll see "verbs" that allow you to do exactly the same kinds of things that MS-DOS and Unix users do in their "batch" and "shell" scripts. Frontier is a much more powerful and better designed scripting system than its character-based predecessors, but the idea is the same. You write scripts in Frontier to automate all the repetitive dragging and clicking you'd do in the Macintosh Finder. You can

customize and simplify the Macintosh OS for users who wouldn't understand a command-line interface.

Why Frontier?

Why did we call this product Frontier? Because it represents a new way of using your Macintosh. Like the pioneers who first set out for the American West, we don't know the limits of Frontier. We love that feeling. It's a major part of why we got into computers and software in the first place. To be right there - at the leading edge of what's happening in the whole world. That's where computers and software are supposed to be.

But over the last few years, the software industry has gotten into a bit of a rut. Lots of same-old-things, more and more feature-laden, heavy products that require a lot of training to learn and use.

If that's the bad news, here's the good news. There's a way around the problem. Let's teach software products how to work together. That's hardly an astounding idea, but it hasn't happened yet. We wondered why.

We imagined a world where a word processor is just a word processor. Use it to write correspondence, reports, manuals, electronic mail. And a database is just a database. Use it to store tabular information, things like names and addresses, inventory data, statistics. They should work together: you should be able to use your database to create and manage an archive of all your correspondence created with your word processor, for example.

All that's needed is a protocol for connecting applications together. Once software products have ports (or connections, or wires, or whatever you want to call them), who does the orchestration? We believe that power should be delivered to the user through a powerful scripting language. That's why we called our company UserLand, and that's why we built Frontier.

The Importance of "Apple Events"

Over the last few years, while both Frontier and System 7 were in development, we consulted with Apple Computer to be sure that their operating system would support our vision for scriptable applications. The resulting technology, known as "Apple Events" is a perfect fit for Frontier script writing.

When you find an application that supports Apple Events, that means you can write Frontier scripts that drive that application. It means that the product is much more useful than competitive products that don't support Apple Events. With Apple Events, each application or utility becomes a toolkit of solved problems for Frontier script writers.

Therefore, Macintosh System 7 and Frontier were made for each other. System 7 makes it possible for applications to be controlled externally. Frontier makes that capability useful for the first time because, for the first time, scripting isn't limited to controlling a single application. As more developers add interapplication Apple Events to their software, Frontier scripts will be able to do more and more. And because Frontier operates outside applications, it can integrate software as components, making your current suite of applications much more useful. It will also create opportunities for smaller, more focused applications and utilities.

Frontier is Unique

Frontier isn't the first user-scripting language to appear on the Macintosh platform. You may have a scripting language built into your database, communication program or hypermedia toolkit.

But Frontier is unique for the following reasons:

Scripting is our only business. Therefore we have been able to invest all our effort in making Frontier the best scripting language it can be.

You learn one language. With Frontier, you won't have to learn a different language to write scripts for each of your programs.

New power. Many applications don't have built-in scripting languages. For the first time you'll be able to write scripts that drive word processing, page layout, graphics, animation, presentation, outlining, file management, electronic mail and utility software products.

Integration. Because Frontier lives outside of any application, you can write scripts that combine the capabilities of several programs, creating functionality that would otherwise be unattainable.

Frontier is different from keyboard macro programs such as CE Software's QuicKeys 2 or Apple's MacroMaker, because Frontier is a full programming language, with logic, looping, sub-scripts, a powerful data storage model called the Object Database and a user-interface design tool called the Menu Bar editor. In fact, Frontier 1.0 includes scripts that let you send messages to QuicKeys, and QuicKeys 2.1 has been extended to allow you to call Frontier scripts from QuicKeys macros.

Use Your Imagination!

As you learn Frontier, imagine what it would be like to have all its scripting power integrated into your word processor, database, spreadsheet, page layout, communications, outliner, C or Pascal compiler, personal information manager, electronic mail program, all your utilities. Technologically, that's absolutely possible.

Many software companies - including the biggest companies with the most widely used products as well as smaller innovative firms - are already putting Apple Event "wires" in their products. But some products are sure to slip through the cracks. If you've ever wanted to script a particular application, now is the time to let the publisher of that product know how important you think scripting is. The more they hear from their customers, the sooner they'll put the wires in, and the better job they'll do.

That's why this is an exciting time for us here at UserLand Software. Finally, we won't be the only voice asking (at times begging!) for scriptable wires in applications. With Frontier shipping, we can ask our users to help us sell the idea of interapplication scripting. Please don't let us down!

The Promise of UserLand

Our major promise to you, our users, is to live up to our company's name. UserLand stands for giving power to users. Power to customize and automate Macintosh software to exactly suit your needs. To erase the bottleneck between you and and the functionality you need.

Specifically, we promise to:

Offer industry-leading technical support, via electronic mail, fax and over the telephone;

Upgrade the product responsively and in a timely manner;

Always appreciate your patronage, and show that we do;

Work with other Macintosh developers to provide a complete base of compatible application software.

All UserLand employees will work to meet these simple objectives. If we ever fail to provide great support, or upgrade our products, or show our appreciation, or fail to help another developer, please let me know personally. By putting these rules in writing, and committing all UserLand employees to them, we hope to continue to deserve your support.

Finally, thanks for giving Frontier a try! We hope you get a lot from it, and we hope you have fun with all the new capabilities it unlocks.

Thanks!

Dave Winer, President
UserLand Software, Inc.

P.S. You can reach me on AppleLink at USERLAND.CEO, on the CompuServe Information Service in the UserLand Forum, or by calling UserLand Software at (415) 325-5700.

P.P.S. Frontier (Aretha) on the WWW.

Contents Page | First Chapter -- **Introduction**

HTML reformatting by Steven Noreyko January 1996

Chapter 1

Introduction

This chapter introduces you to this Frontier User Guide. It describes who it and the Frontier product are intended for, the purpose of this book, and how it is organized. This chapter also explains the typographical conventions used in the book. It provides some ideas how to proceed through this manual depending on your background and interests. It tells you how to install Frontier on your system. Finally, it tells you where to look for more information.

An Overview of the User Guide

This Frontier User Guide is aimed at people who want to use Frontier to manage their Macintosh environments and systems more effectively. There are probably as many different types of readers and users of Frontier as there are Macintosh owners. But Frontier is aimed specifically at two categories of users:

- programmers, including HyperTalk scripters, spreadsheet macro writers, and professional or hobbyist programmers working in traditional programming languages like C and Pascal, whether for commercial distribution or in-house use
- interested users, by which we mean people who care enough about how their systems work that they want to take them apart and tinker with them, making them behave just a little differently from the next person's

If you fall into one of those categories, then Frontier and this manual are for you.

Purpose of the Book

The purpose of this book is to provide you with as complete an explanation as possible of how to get started with, use, take advantage of, and ultimately master the Frontier environment and its UserTalk programming language.

One thing this book does not attempt to do is to teach you programming skills. You should already be basically familiar with concepts such as loops, conditional processing (if-then-else constructs), and variables.

This book also assumes that you understand how to use your Macintosh reasonably effectively. You should know not only how to use the mouse, navigate menus and deal with windows, but you should also be familiar with how the Finder works, how files and folders relate to each other, and how System 7 operates.

If you have this background, you should become quite successful at learning and using Frontier and UserTalk.

How the Book is Organized

This manual is organized into 10 chapters and one appendix.

Following this introductory chapter, you'll dive right into Frontier in Chapter 2 with an explorer's guide to the inner workings and power of the product.

Then in Chapter 3 you'll take a look at one of Frontier's most accessible and useful aspects: desktop scripts. These scripts are written in Frontier's UserTalk language and enable you to undertake a wide variety of tasks on your desktop, including some you may have thought were impossible with graphical user interface-based environments.

Chapter 4 explains all of the menus, windows, and other user interface elements that make up the Frontier environment, explaining every option so that you have a single source to refer to for information about how to accomplish tasks in Frontier.

In Chapter 5, the focus is on the powerful core of Frontier, its Object Database. Frontier stores all of its variables, information it needs while running, scripts, and its entire language here. You'll come to appreciate how to use the Object Database and make it truly your own.

The mechanics and concepts involved in writing UserTalk scripts are the focal point of Chapter 6. You'll learn how to create, edit, store, comment, format, compile, run, and debug scripts, and how to attach them to Frontier menus for ease of execution.

Chapter 7 provides you with an overview of the UserTalk scripting language built into Frontier. It describes the scope of the language, tells you where to go for on-line documentation, and discusses variables, operators, datatypes and other language-specific matters.

The final three chapters contain examples of programming UserTalk. Chapter 8 focuses on building applications that live entirely within Frontier. In Chapter 9, we turn our attention to the outside world, focusing on the Finder and on the design and creation of desktop scripts. Finally, Chapter 10 concentrates on scripting for interapplication communication (IAC), including how to package the final product for delivery to other Frontier users.

Appendix A is a glossary of terms used in this and the other manuals in the Frontier documentation set.

Finding Your Way Around

Depending on your background, you might approach this manual and Frontier somewhat differently from other users.

If you are an experienced professional programmer with a knowledge of C or Pascal or another modern programming language, you will probably find yourself best served by skimming Chapters 2 and 3, reading Chapter 4 more than cursorily, and concentrating your attention on Chapters 5-7. Then work through one or more of the examples in Chapters 8-10.

If your experience is with less traditional programming and scripting tools like HyperTalk or

a spreadsheet macro language, then we suggest you work through Chapter 2, skim Chapter 3, read through Chapters 4 and 5 without being too concerned about understanding all of their content at this point, and concentrate on Chapters 6 and 7. As you gain experience, you'll undoubtedly want to return to Chapter 5 for background. Work through all of the examples in Chapters 8 and 9 and then try some things on your own before proceeding with Chapter 10.

Typographical Conventions

In this manual, verbs that are part of the UserTalk scripting language are shown in Courier font, as in file.copy.

Many verbs require parameters. These are printed in boldfaced italics when they are being described as part of the formal syntax of the verb in the UserTalk Reference Guide, and in italics on subsequent references in that guide and in all uses in this manual.

Example code is displayed in Courier font. In the UserTalk Reference Guide, examples are intended to be typed into the Quick Script window, except where indicated. The return value from a verb is also indicated in Courier font.

System Requirements

Frontier requires a Macintosh computer system capable of running System 7.0, at least one megabyte of RAM, a hard disk, and System 7.0 or higher. It is compatible with file-sharing under AppleShare with System 7.0 and prints on all LaserWriter and ImageWriter-compatible printers.

Before Proceeding

Complete and mail in the registration card located in the Customer Service and Registration Information card in the package. Retain the front portion of the card. It contains your registration number, which will identify you as a registered customer of Frontier, and will entitle you to free technical support, software updates, the latest details on products newly compatible with Frontier, and specially priced upgrades. You might consider making a note of this number inside your Frontier User Guide, just in case you lose your card.

Installing Frontier

Refer to the sheet "Installation Instructions" in the package for details on installing Frontier.

If You Want Assistance

UserLand Software can assist you in a number of ways, should you have questions about working with Frontier.

On-line help with DocServer. See Chapter 6 for details on how to use DocServer.

AppleLink Discussion Board, inside the Third Parties Folder on AppleLink.

AppleLink support address userland.dts@applelink.apple.com.

UserLand support forum on the CompuServe information service.

UserLand folder on the America Online information service.

Telephone support at 415/325-5700 or via facsimile at 415/325-9829.

See the Customer Service and Registration Information card inside your package for more details.

Terminology

Any product as rich and robust as Frontier necessarily involves the use of a number of new terms and concepts. These ideas are introduced at appropriate points in the book; however, if you encounter a term that is either unfamiliar to you, or is used in a way you haven't seen before, turn to the Glossary in Appendix A. Chances are, you'll find it explained there.

HTML reformatting by Steven Noreyko January 1996

Chapter 2

An Explorer's Guide to Frontier

This chapter presents a brief tour of UserLand Frontier using lots of screen shots and a limited amount of commentary. Its purpose is to help you get acquainted with the elements and capabilities of Frontier so that you have a framework within which to understand the rest of this book.

Put on your explorer's hat. We can't tell you everything about the software in this tour. But don't be afraid to take detours or tinker with aspects of the product that you find interesting. You can quit without saving changes, or use the Revert command on the File menu, which is a good safety valve in case you make a bunch of changes and then can't figure out how to get back where you started. It's a good idea, too, to use the Backup command in the UserLand menu to create a backup of the primary Frontier file before you make any changes.

This chapter begins with a brief discussion of the several types of files that make up Frontier's environment. Then it moves through the main components of Frontier, in the following order:

- the Main Window
- the Quick Script window
- the Object Database and Table Windows (where we'll also discuss variables briefly)
- word processing windows
- outline windows
- the Menubar Window (where we'll also see where scripts connected to the Menubar are stored)

The chapter concludes with an example of a typical, though abbreviated, Frontier session, where you'll create a new object in the database, write a short script to make use of it, define a menu, and attach a script to a menu item to work with the newly created object.

The Frontier Files

Stripped to its barest bones, Frontier consists of just two files: the application and a file called Frontier.root. The Frontier.root file which you'll come to think of as simply "the root" is where Frontier stores all of its information. Your Frontier scripts and their associated data will also be stored in this file. In addition, you may find the Object Database that is at the heart of this file to be a handy place to keep lots of other information. We'll have much more to say on this subject in Chapter 5, which is devoted entirely to the Object Database.

As you work with Frontier, you will probably create or use at least two other kinds of files: desktop script files and import/export files.

A desktop script file is similar to a Macintosh stand-alone application except that it can't run without Frontier. There are a number of these files on the distribution disks on which Frontier is shipped and you may find yourself creating others.

Import/export files contain Frontier objects which have been exported from Frontier for later loading into a root file.

These two file types have identical icons. If you double-click on a desktop script icon, it will launch Frontier (if Frontier is not already running) and then execute the script(s) contained in the file. If you double-click an exported Frontier icon file, it will launch Frontier (unless Frontier is already running) and then it will ask you where you want to store the object you are about to import.

If you hold down the Command key while you double-click on either type of file, and keep it held down until Frontier has opened the file, Frontier opens the script editor for the object rather than running the script. (In the case of an import/export file, the script opened by Frontier is saved with all exported objects.)

The Main Window

When you boot Frontier you'll see a small narrow window, with a flag, a close box and a resize box (see Figure 2-1).

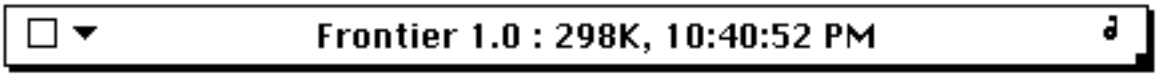


Figure 2-1. Frontier's Main Window

This window is called the Main Window. When you click on the flag at the right end of the window, the window gets taller, revealing four previously hidden buttons (see Figure 2-2). If you save Frontier with the window in this condition, it will open this way next time you launch Frontier.

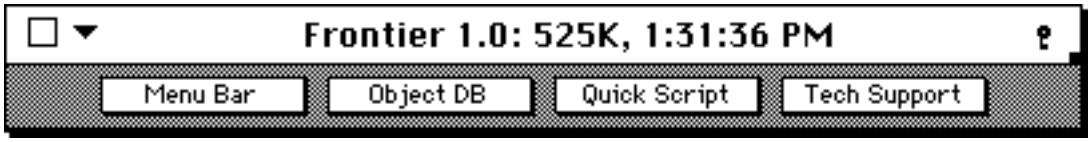


Figure 2-2. Main Window Expanded to Show Buttons

Flipping down the flag is like lifting the hood on the car: you don't have to lift the hood in order to use Frontier. But if you want to change anything, or add new commands, or just see how things work, flipping down the flag is one way of beginning the process.

Each of the four buttons in the expanded Main Window zooms out a window. "Menu Bar" gets you to the menu editor window. That's where you edit or change the commands in the menubar, or the scripts linked into them. "Object DB" zooms out the top level of the Frontier Object Database. More about that later in this chapter and in Chapter 5. "Quick Script" zooms out a window where you can type short commands to Frontier. "Tech Support" zooms out a help window, with information about getting more information or help. This button's behavior is entirely up to you, the Frontier script writer.

Normally, the Main Window in Frontier displays version information about Frontier, the amount of memory available, and the current time. You can also place information into this window directly.

As a Frontier script writer, you can lock the flag in the Frontier window in the "up" position. This would make it somewhat harder for unauthorized users to look at scripts or examine the contents of the Object Database.

Contents Page | Next Section -- **Quick Scripts**

Chapter 2: continued

Quick Scripts

First, let's look at the "Quick Script" button, and the window it zooms, the Quick Script window (see Figure 2-3).

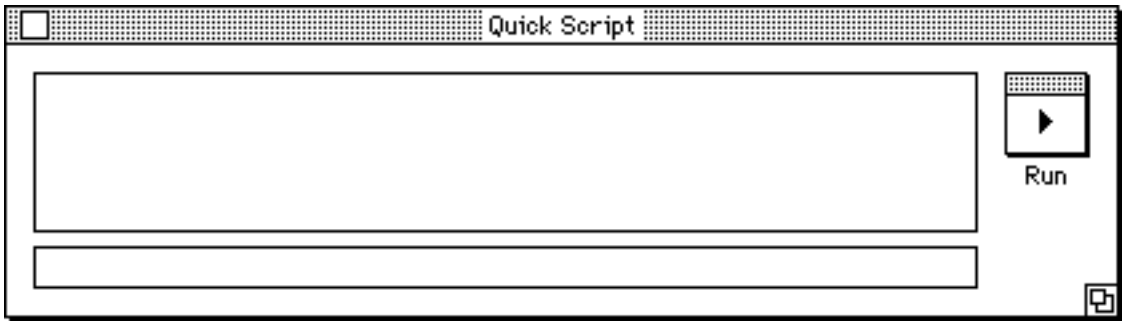


Figure 2-3. The Quick Script Window

To use the Quick Script window, just type an expression into the editing area (the larger rectangle at the top of the window) and either click on the "Run" button or press the Enter key (not the Return key). In Figure 2-4, the user has typed an arithmetic expression into the window and executed it. The result is shown in the message area at the bottom of the window.

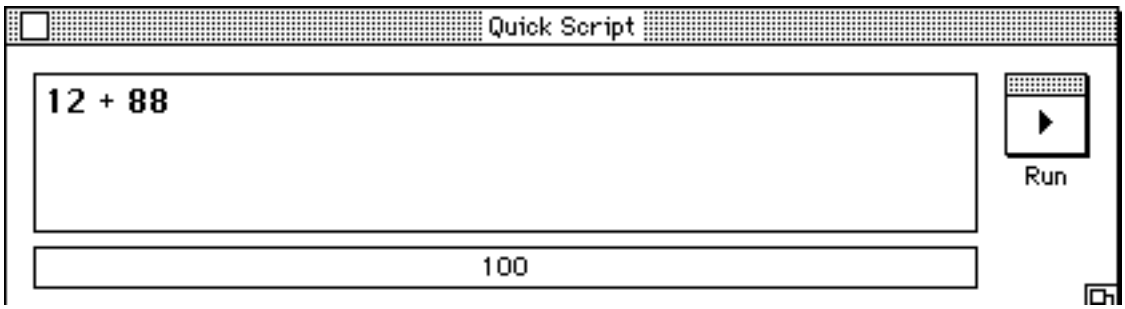


Figure 2-4. Sample Quick Script Expression

When you type into the Quick Script window you're actually writing and running a one-line UserTalk script. To prove it, try typing something that makes no sense, as in Figure 2-5.

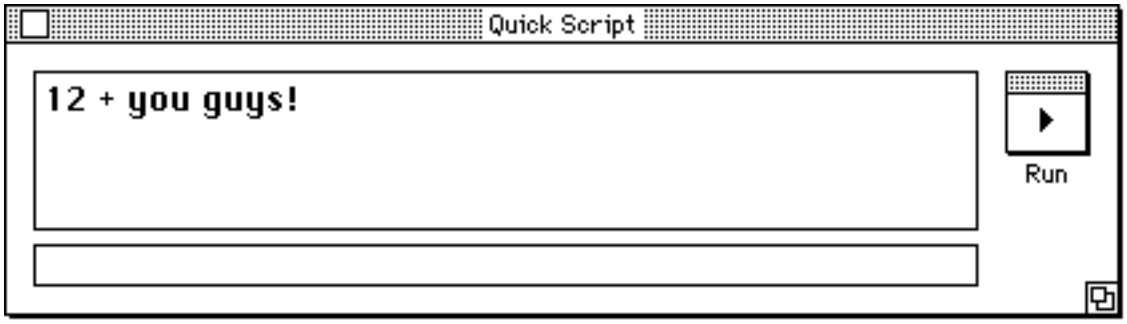


Figure 2-5. Nonsense Entry in Quick Script Window

Frontier responds with an error window with a message explaining why it didn't like what you typed (see Figure 2-6).

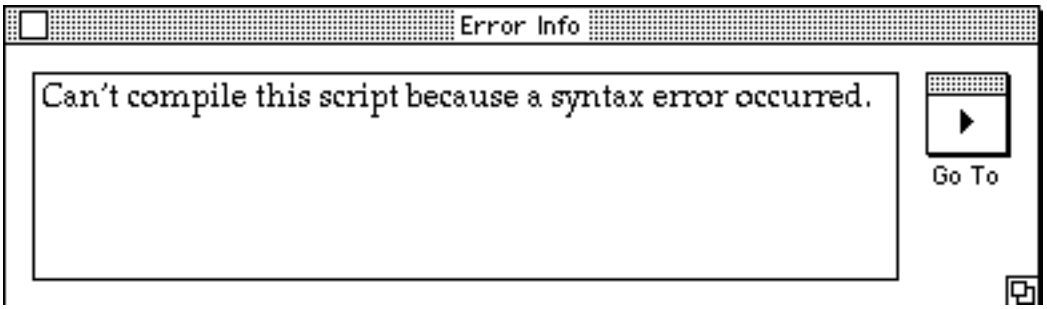


Figure 2-6. Typical Error Message

Type the script shown in Figure 2-7 into the Quick Script window and press the Enter key:

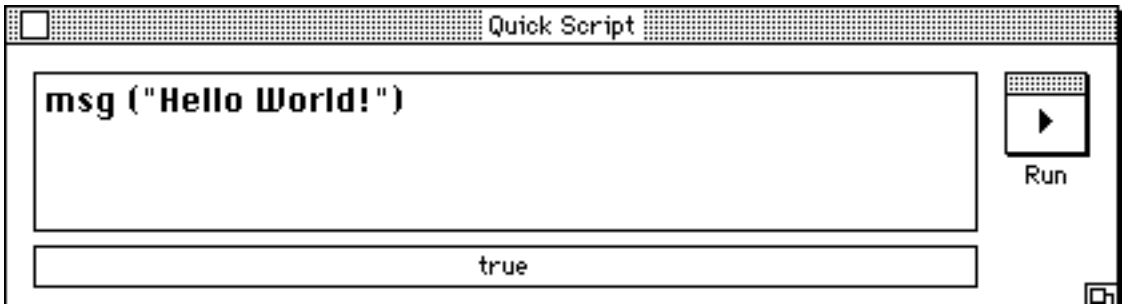


Figure 2-7. One-Line Script in Quick Script Window

Now, check out the Main Window. As you can see (Figure 2-8), the text in parentheses in the small script you just typed appears in the Main Window in place of the status information that was there originally. (You can return the Main Window to its normal task of showing you the present time and memory available simply by selecting it.)

Figure 2-8. Message Displayed in Main Window

The first part of the script you typed in Figure 2-7 the word msg is a Frontier verb. It requires one parameter, a string of characters surrounded by double quotes. The msg verb

displays the string parameter in the Main Window. All Frontier verbs return a value when they execute unless they produce an error message. Unless you indicate otherwise, this result will be displayed in the message area of the Quick Script window for scripts entered and executed in that window. As you can see in Figure 2-7, msg returned a value of "true," indicating that the verb executed successfully. (In fact, msg always returns "true" because it can't fail.)

The Frontier scripting language UserTalk implements hundreds of verbs. They all follow the same pattern. Verbs have names, take parameters, and return values. (Full documentation on the UserTalk verbs can be found in the UserTalk Reference Guide, a companion volume to this one.)

Some verb names are structured. Figure 2-9 presents an example of such a verb. The name of the verb is string.upper. This name tells Frontier to look in a table of verbs in the Object Database named string, for a verb named upper. As you can see, string.upper takes a string as a parameter, and returns the same string converted to uppercase.

Figure 2-9. Frontier Verb With Structured Name

Chapter 2: continued

Variables, the Object Database, and Tables

All the examples so far have simply carried out computations and displayed their values in the Quick Script window. You can also store values in variables so they can be used later. Figure 2-10 shows an example of a script that places a string into a variable named examples.greeting.

Figure 2-10. Sample Variable Assignment

We can then use the new variable in another Frontier script, as you can see from Figure 2-11.

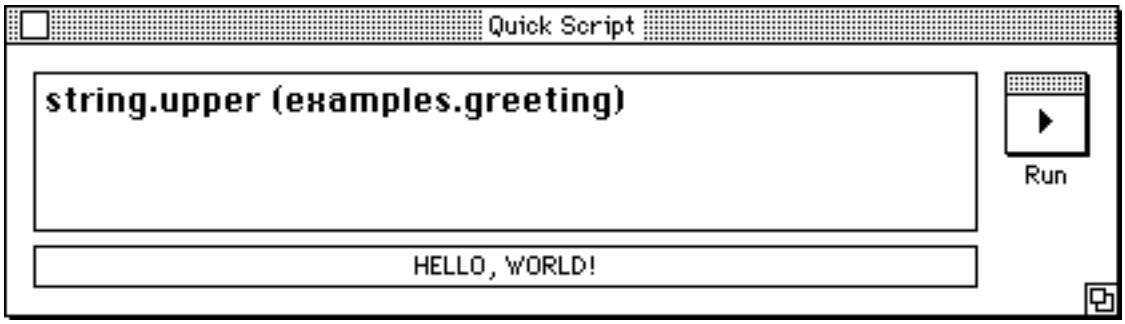


Figure 2-11. Use of a Sample Variable

That's how you store values in variables. Variable values are stored permanently in Frontier's Object Database. Next time you start Frontier there will be a variable named examples.greeting and its value will be "Hello World!"

You can easily browse through all the stored values and change them interactively by opening the Object Database. There are two ways to do this:

Open the Main Window, make sure the buttons are showing, and click on the "Object DB" button.

Select "Object Database" from Frontier's UserLand menu (or use Command-T, the Command-key equivalent of this menu choice).

In either case, you will see the Object Database table open into a window that looks something like Figure 2-12.

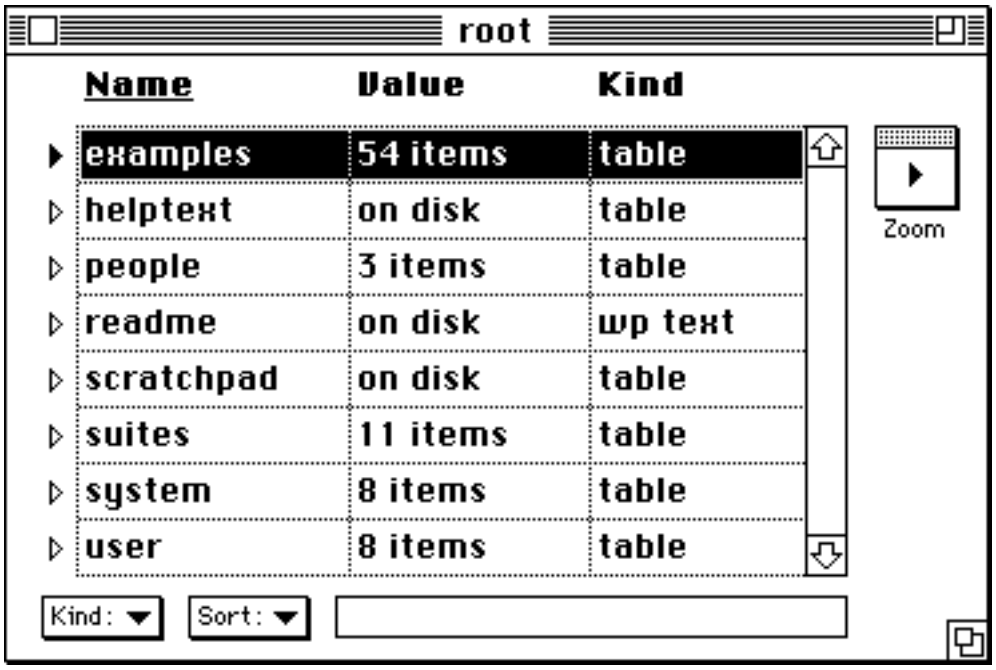


Figure 2-12. Top Level of Object Database

This table is the place where all variables and values are stored. You may think "Object Database" is too-fancy a name for what's going on here. But stay with it; you'll see soon why it's a database that stores objects.

Select "Open Examples" from the Custom menu. (You can also type the letter "e" while the root window is frontmost, locate the entry for the examples table, and double-click its item marker.) The window that opens should look something like Figure 2-13, although it may be a different size and may be scrolled to a different position.

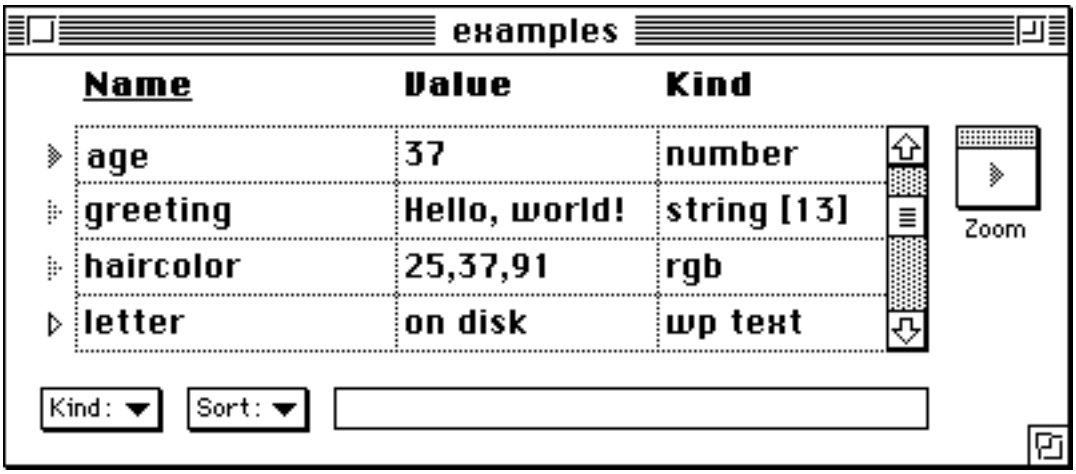


Figure 2-13. Inside the Examples Table

This is where examples.greeting is stored. The table is named examples and the entry is called greeting. To prove this, change the value in the Quick Script window by typing the script shown in Figure 2-14 and executing it.

Figure 2-14. A New Value for examples.greeting

As you can see (Figure 2-15), the value of the variable examples.greeting was changed by the script you just ran from the Quick Script window.

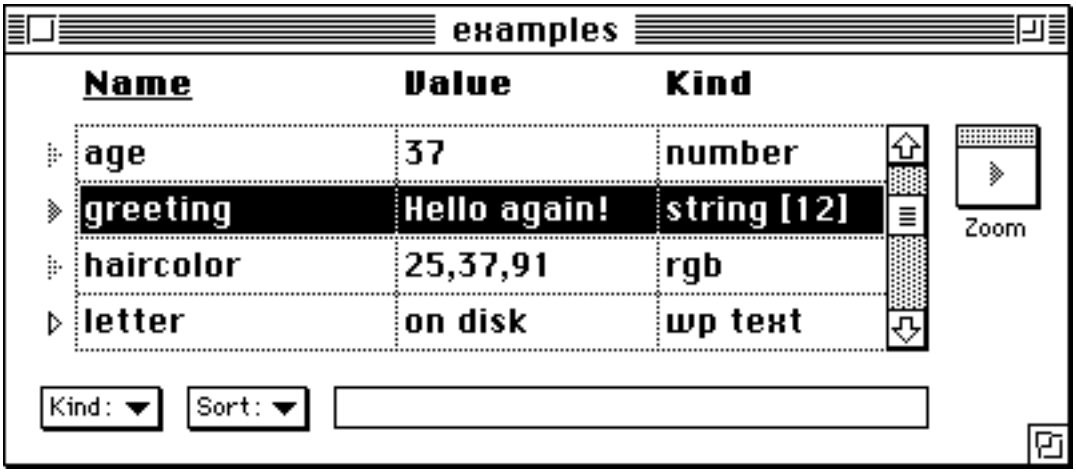


Figure 2-15. Variable Value Modified

There are a number of ways you can create variables. You will probably create them most often either from a UserTalk script or interactively in a table editor.

Try typing "examples.x = 12" (without the quotation marks, of course) in the Quick Script window. Note that there's a new value in the examples table called "x" and that it has a value of 12.

Creating a new variable interactively is a lot like adding information to a spreadsheet. Position the cursor anywhere in the table where you wish to store the new variable. Hold down the Command key and press Return. This opens a new entry in the table. Type the variable's name in the Name column. Tab to the Value column and enter a value for the variable. Optionally, you may wish or need to select one of the items in the "Kind" popup menu, although most of the time Frontier handles this for you. You've created a new entry in the table.

There are other ways of creating new table entries and assigning them both a kind and a value. We'll discuss this subject in more detail in Chapters 4 and 5.

Frontier offers two different ways of editing the contents of objects in the Object Database. Some objects are edited "in place" in the table. You can tell these items because the item marker before their name is gray rather than a continuously outlined triangular shape. If the mouse cursor becomes an I-beam when you move it over the Value column of a variable, this is another indication you are dealing with a variable that can be edited only in the table window. Only the simplest types of information (such as numbers and strings) can be edited in the table directly.

All other objects can only be edited in their own windows. Double-clicking on the item marker to the left of the entry's name in the table will open its editing window.

Each object in the Object Database is either a table or some other type. To see all the different types supported by Frontier, click on the Kind popup in the lower left corner of a table-editing window. An object can be as small as a number or a string, or as large as a script, word processing document or outline.

Some types can be converted to other types. In the examples table, look at the value of a variable called "green." Note that it is a number. Click on the item marker next to green and select "String" from the "Kind" popup. Notice that green is now a string of length 2 (that's what string [2] means). Convert it back to a number by selecting "Number" from the "Kind" popup.

You can resize the window the usual way. Frontier remembers the size and positions of all its windows from one session to the next.

You can change the widths of columns in a table editor. Carefully move the mouse over the border lines between columns. When the cursor changes to a pair of vertical "rails," press the mouse button down and drag the border line. Let up on the mouse whenever you like. The other two columns automatically resize to fill the remaining space.

Tables can contain sub-tables, and sub-sub-tables and so on, up to 25 levels deep. To refer to something that's nested in a table, put a dot between the name and its table's name, as in examples.greeting. The name examples.subTable1.message refers to an object named "message" in the sub-table called subTable1, which in turn is stored in the examples table.

Note, too, that when a table is open as the frontmost window in Frontier, a special menu called (logically enough) "Table" appears on the menubar. This menu contains many commands that you'll find useful in working with tables. Its commands were created in UserTalk and you can modify its behavior if you're adventuresome. The menu's basic usage is discussed in Chapter 4; we'll talk about menubars and their scripts later in this chapter.

Chapter 2: continued

Word Processing Windows

To facilitate the storage and use of fully formatted text objects in its Object Database, Frontier includes a fully scriptable word processor. You can use this capability to create documents in Frontier and to format text documents you bring into Frontier from outside the Frontier environment.

Open the word processing object called examples.letter by double-clicking on the item marker next to its name in the examples table. You should see something like Figure 2-16. The content and scrolled position of the document, as well as the window's size, might be different on your system.

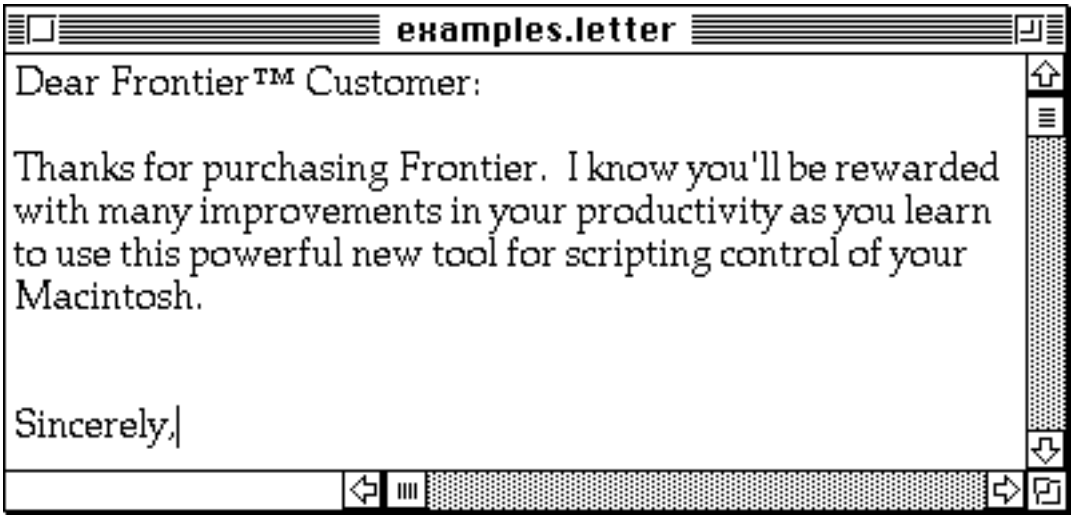


Figure 2-16. Sample Word Processing Text Window

You will also notice that a new menu appears in the menubar. It's called "WP" and it appears whenever a word processing text object's editing window is frontmost in Frontier.

The WP menu lets you toggle the ruler on and off or time-stamp a document (creating a new line at the insertion point with the present date and time along with your initials). With the ruler symbols, you can set and clear tabs and format text justification. With commands on the Edit menu or their keyboard equivalents, you can change fonts and styles of some or all of the text in a word processing text object.

If you've used any popular Macintosh word processor, using the word processing editing window will be comfortable and familiar. The crucial difference is that the Frontier word processor is fully scriptable, supported by more than three dozen UserTalk verbs that operate on word-processed text.

Outline Windows

One of the most important aspects of Frontier is its exploitation of the outline metaphor. As you'll see, all UserTalk scripting is done in an outline form (except when you use the Quick Script window). Menubars are also maintained and edited as outlines, as we'll see in the next section.

But you can use Frontier outlines for storing and clearly organizing information of any type. To see a typical outline-editing window, locate an entry in the examples table called Universe and open it in the usual way. Figure 2-17 shows you approximately what this window should look like.

Figure 2-17. Typical Frontier Outline

As you can see, this is a fairly typical outline format (though its contents are admittedly something less than gripping.) It is shown in its "fully expanded" mode. Each line in an outline is referred to as a heading or sub-heading. The headings at the highest level of the outline are referred to as "summits." (Unlike some other commercial outliners, Frontier's built-in outliner permits you to have more than one summit at the leftmost position in the outline.)

In the sample outline, double-click on the item marker next to the label "North America." The outline now looks like Figure 2-18.

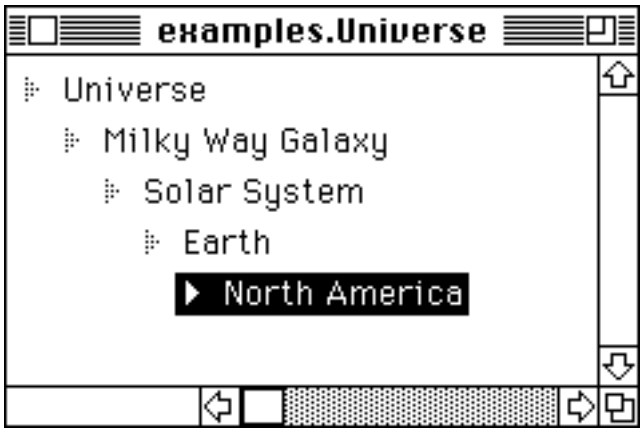


Figure 2-18. Partially Collapsed Sample Outline

You have now collapsed all of the sub-headings under the one on which you double-clicked. Notice that its item marker is solidly black. (You may have to move the bar cursor that is highlighting the line to see this clearly.)

As you might have noticed, there is a new menu in the menubar called "Outline." Select the "Full Expand" option from that menu and your outline returns to its previous state. Close the outline window.

We will have much more to say about using outlines in Frontier throughout this manual, but particularly in Chapter 4, when we discuss the Frontier environment.

Contents Page | Previous Section | Next Section -- **The Menubar Window and Scripts**

Chapter 2: continued

The Menubar Window and Scripts

Frontier's menubar has four permanent menus: Apple, File, Edit, and Window. The rest of the menus are open for you to create, delete, reorganize, or modify. All editing of the menubar is done through the menubar window. There are several ways to open that window:

- click on the Menubar button in the Main Window
- select "Menubar" from the UserLand menu
- type Command-M for the keyboard equivalent of the menu selection
- hold down the Option key while opening any of the menus other than the four permanent ones defined above

When you do open this window, it will look something like Figure 2-19 (again, allowing for the possibility that it may appear different on your screen for any of a number of reasons).

Just as spreadsheets allow you to work with rows and columns of numbers, and word processors are good at editing and revising letters and reports, the menu editor in Frontier lets you edit the menubar as if it were a document.

The menu editor works just like an outline. Double-click on the item marker to the left of each line to expand and collapse sub-heads. Switch in and out of text mode (so you can edit individual menu entries) by pressing the Enter key. The arrow keys move the cursor.

Any changes you make in the outline are automatically and immediately reflected in the menubar.

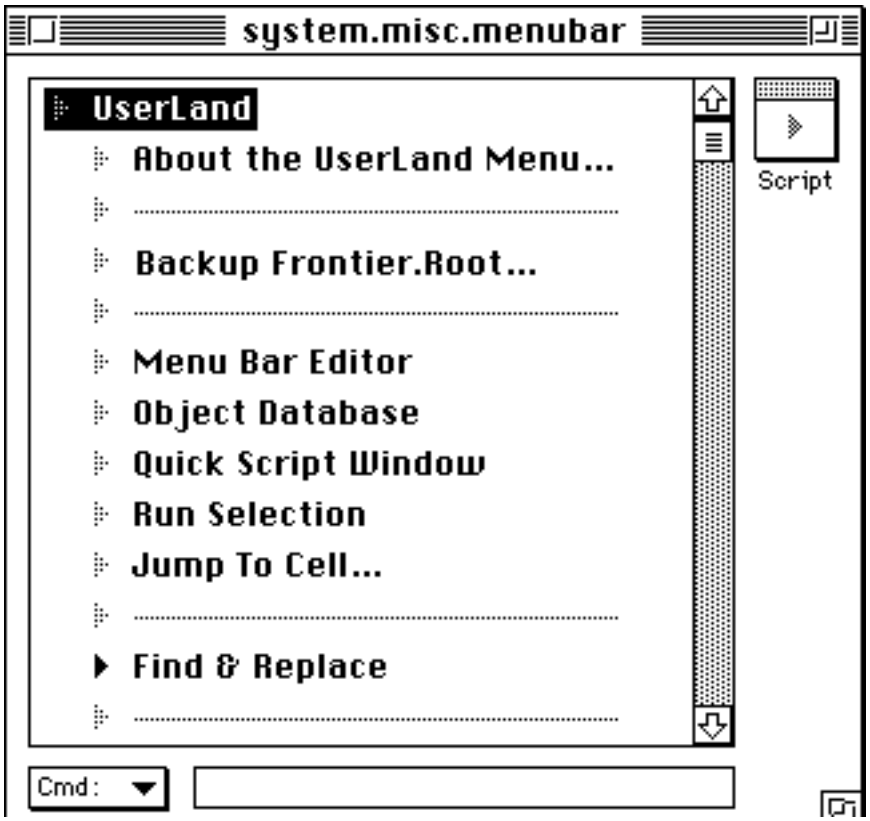


Figure 2-19. The Menubar Editing Window

Any menu item can have a script linked to it. To create, edit, or examine the script activated by a menu item, select the item and then click on the "Script" button in the upper right corner of the menubar editor's window. If the Custom Menu isn't fully expanded and visible, scroll to it in the menubar editor and then double-click on the item marker immediately to the left of the headline "Custom." Locate the menu item called "Fun Stuff." If it isn't open, double-click on the item marker immediately to its left. Figure 2-20 shows what this part of the menubar outline should look like.

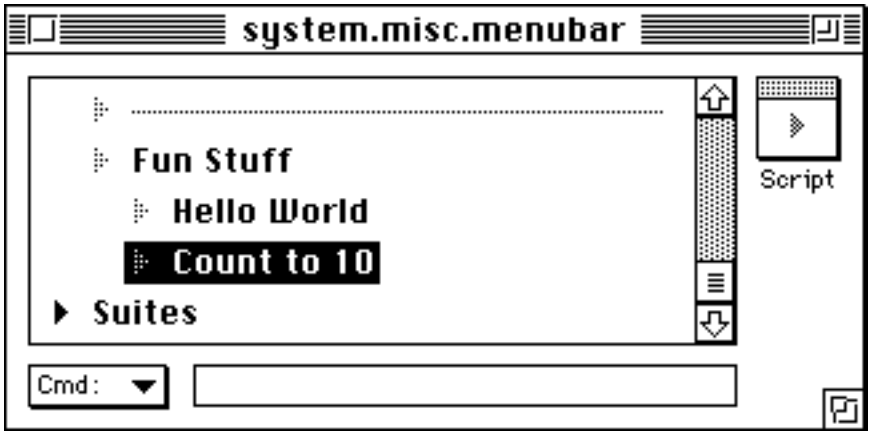


Figure 2-20. Partial Custom Menu Outline

Pull down the Custom menu and have a look at the "Fun Stuff" menu. There should be a one-to-one correspondence between what you see in the menubar and what you see in this window. Put the cursor on "Hello World" in the menubar editing window and click on the

Script button in the upper right corner. This zooms out the script window linked to the "Hello World" entry in the menu, as shown in Figure 2-21.

Figure 2-21. Script Associated with Menu Entry

You can run the script by clicking on the "Run" button, or by selecting the item from the menu. Clicking on the "Run" button executes the script to completion. Clicking on the "Debug" button gives you all the debugging options that a script writer needs (see Figure 2-22).

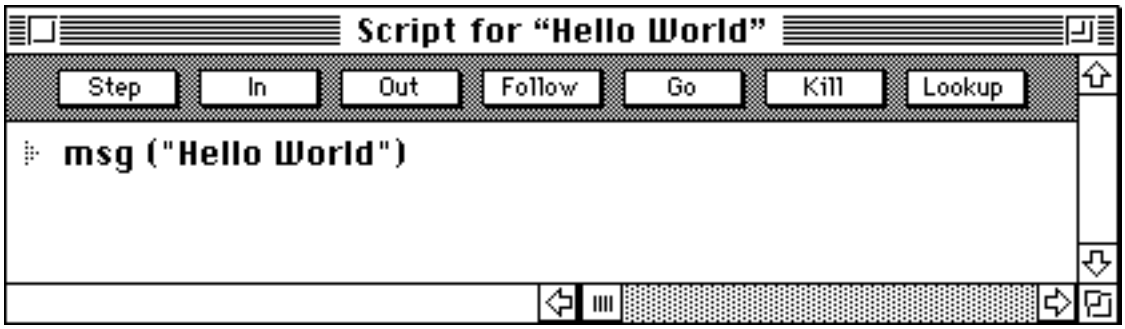


Figure 2-22. Script-Editing Window in Debug Mode

Click on the "Go" button to run the program. It will come as no surprise, given our earlier experiment in the Quick Script window, that the main window now has a message in it saying "Hello World."

Bring the menubar window to the front, move the cursor to "Count to 10" and click on the Script button. This is a simple loop that counts to 10 in the main window (see Listing 2-1).

```
counter = 0
loop
    counter = counter + 1
    if counter > 10
        return ()
    msg (counter)
```

Listing 2-1

To see what it does (just in case it's not obvious) and to show you the direct connection between what's in this script and the menu itself, choose "Count to 10" from the Custom

Menu's "Fun Stuff" sub-menu. Notice that the numbers one through 10 appear in the Main Window.

Now click on "Run" and watch it work.

First it initializes a temporary variable, named counter, to 0. Then it loops, adding one to counter until it's greater than 10. Every time through the loop it uses the msg verb to display the value of counter in the Main Window.

The same effect could have been achieved in the much shorter script shown in Listing 2-2.

```
for i = 1 to 10
    msg (i)
```

Listing 2-2

Contents Page | Previous Section | Next Section -- **A Sample Frontier Scripting Session**

Chapter 2: continued

A Sample Frontier Scripting Session

We know how eager you are to get your hands on Frontier and actually do something. So let's take a few moments to walk through an abbreviated but realistic scripting session with Frontier. Don't worry if, along the way, some of the concepts seem a little fuzzy; later chapters will clarify the ideas that we will only have time to touch on here.

Most Macintosh owners we know suffer from hard disk clutter. They accumulate sizable numbers of copies of files they use frequently, temporary files that never seem to go away, unintentional backups of various kinds, and the kind of digital clutter that turns a pristine hard disk into someone's attic. Often, they don't realize the problem they've created until they save a document they've been working on for the past two hours without saving it and get the dreaded "disk full" message.

A script that would let you know what percentage of your disk is still available would help you avoid such situations. So let's build one in the UserTalk language. Fortunately, UserTalk includes a couple of verbs that are going to make this script quite easy to write.

In true Frontier fashion, we'll include in this script the idea that the user should be able to decide what kind of safety margin to impose on disk-full situations. So the first thing we'll do is set up a variable to contain the safety margin. When you ran Frontier for the first time, you were asked for your name and initials. Frontier created a table called people.ME, but it put your initials in where the "ME" are in that table name. We'll use ME for this discussion, but remember that we're talking about your people table.

Creating a Safety Margin Entry

Open the people.ME table by choosing "Open people.ME" from the Custom menu. Assuming this is your first time to open this table, it should look like Figure 2-23.

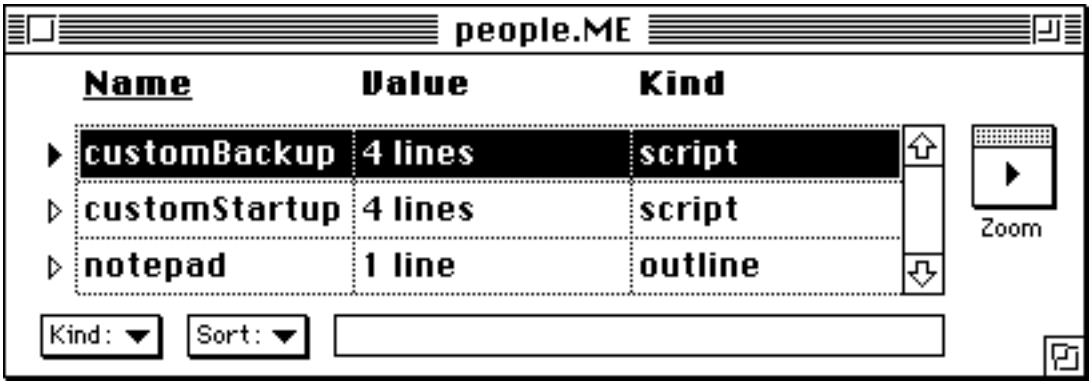


Figure 2-23. Newly Initialized people.ME Table

Press Command-Return to create a new table entry. Frontier leaves you in a position to name the newly created object. Call it "diskSafetyMargin." (Note that we've created the variable as if it were one word long. Frontier permits the use of multi-word variable names but we don't want to get into the slight complication in scripting that using such variables introduces.)

Tab to the "Value" column and type the number 10. Press the Tab key or simply click in another part of the table to complete your entry. Notice that Frontier looks at the value you entered and determines that you want to store a number in this cell, so it provides the word "Number" in the "Kind" column of your new entry.

Now press Enter to leave text editing mode and return to table mode. The entire row where your new variable is stored will be selected, as in Figure 2-24.

Don't worry that the new variable isn't sorted correctly with the rest of the entries. When you close this window and re-open it, Frontier will take care of that for you. (If you're the kind of person who really worries about such things, click on the "Sort" popup menu and choose "by Name" from the resulting popup. You can also just click on the "Name" label in the Table window. Frontier will restore sorted order to the table.)

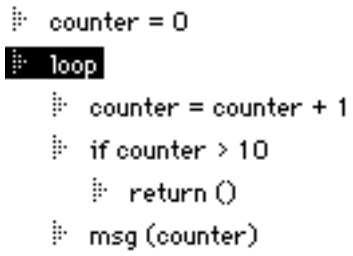


Figure 2-24. Completed Variable Entry

Writing the Script, Step 1

Now we have a variable against which to compare our disk's percentage of free space, so let's write the script that will do the comparison.

Press Command-Return in the people.ME table. In the "Name" field, type "diskPctFree." Pick "Script" from the "Kind" popup menu. Now double-click on the item marker next to this newly created script entry. Frontier will zoom open a script-editing window.

The first thing we have to do is help Frontier identify the disk we want it to examine. The file.getSystemDisk verb will tell Frontier the name of the disk from which we boot our Macintosh. Type the following line exactly as it appears here:

```
f = file.getSystemDisk ()
```

Writing the Script, Step 2

Now we want our script to calculate the percentage of free space on this hard disk. Press Return. Frontier gives you another item marker in the script-editing window. Type the following line into the script-editing window, exactly as it appears here:

```
pctFree = double (file.freeSpaceOnVolume (f)) / file.volumeSize (f) * 100
```

This statement uses three Frontier verbs, which are, briefly:

- double, which simply makes sure that the value returned by the next verb is in a format we can use to calculate a decimal value
- file.freeSpaceOnVolume, which produces a number indicating the number of bytes of free space on the volume called "f" (which, of course, we've just defined as our system volume)
- file.volumeSize, which produces a number indicating the total capacity of the volume in bytes

By dividing these two numbers, we get a result like ".2187." So we multiply the result by 100 to get it into a format like "21.87." This is the percentage of free space on the disk.

Writing the Script, Step 3

Next, we compare the value of this calculation against the safety margin value we set earlier. If the disk's free space is within limits, we display one message; if not, we display another. Here are the rest of the lines in the script:

```
if pctFree < diskSafetyMargin
    msg ("Time to clean house on your disk!")
else
    msg ("There's enough room on your disk!")
```

Notice the indentation of the second and fourth lines. This is important. After you type the first of the above lines (which will become the third line in your script), press Return and then the Tab key to indent one level. Then when you enter that line and press Return, be sure to press Shift-Tab to return to the top level of the outline again. The same process will format the third and fourth lines correctly, as shown.

Compiling and Error-Checking the Script

Click on the "Compile" button in the script-editing window. If any errors result, click on the "Go To" button in the error window (see Figure 2-25). Frontier will take you to the point where the error occurred. Check your script to make sure it exactly matches what we have shown you.

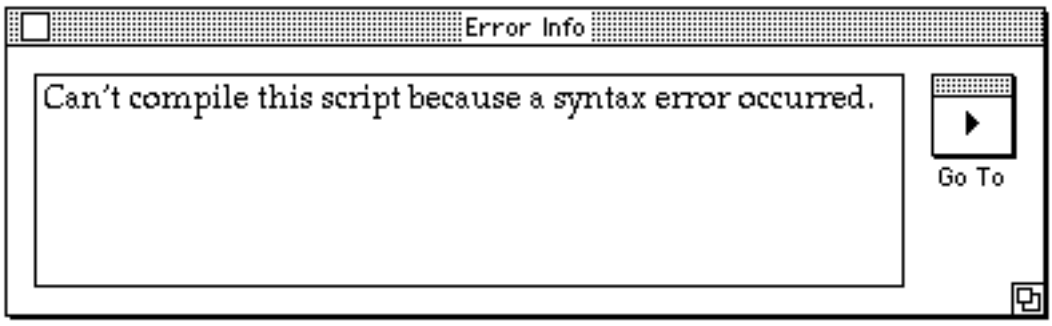


Figure 2-25. Error Information Window and "Go To" Button

Here's the entire script for reference purposes:

```
f = file.getSystemDisk ()
pctFree = double (file.freeSpaceOnVolume (f)) / file.volumeSize (f) * 100
if pctFree < diskSafetyMargin
    msg ("Time to clean house on your disk!")
else
    msg ("There's enough room on your disk!")
```

Once you've successfully compiled your script, click on the "Run" button. One of the two messages should appear in your Main Window.

Congratulations! You've written your first Frontier script and it's a pretty useful little critter besides!

Connecting to a Menu

This new script seems somewhat inconvenient to use because you have to open your people.ME table, find the script, open it, and click on "Run." Actually, there are two other, somewhat faster ways to run the script.

You can type the script name into the Quick Script window, followed by a pair of parentheses, and it will run (see Figure 2-26).

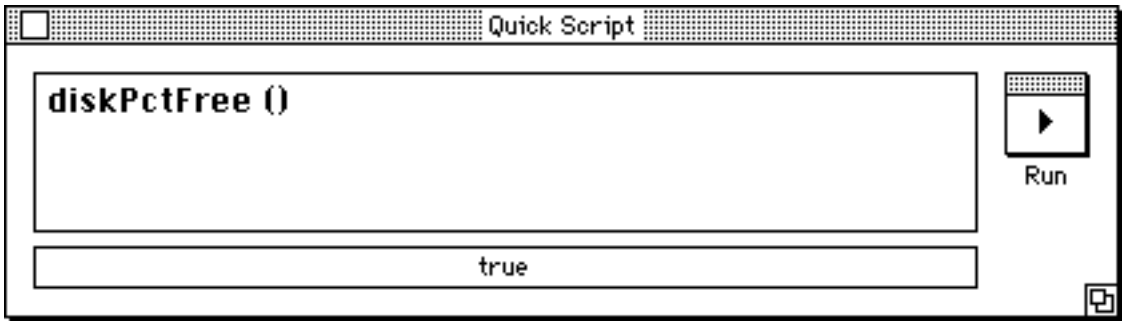


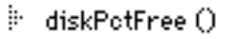
Figure 2-26. Running Your Script from the Quick Script window

You can also position the cursor anywhere in the line where the script is stored in the people.ME table and press Command-slash to run the script.

But both of these methods, while more convenient than opening the script itself, still require you to do more work than you should have to. After all, it's your script, right? So let's attach it to a new menu in the Custom menu.

Hold down the Option key and pull down the Custom menu. With the Option key still held down, select "Open Examples" and release the mouse button. When the menubar editor opens, the bar cursor (highlight) should be on "Open Examples." Press Return. Frontier gives you a new menu entry with an item marker. Type the new menu's name. We'll use "Check My Disk" but you can call it anything you want.

Once you've named the menu entry, click on the "Script" button. This will open an empty script-editing window. Type the following line of UserTalk code exactly as it appears here:



Close the script-editing window and the menubar editor window. (This might be a good time to save your root file if you want to keep this menu item around. Press Command-S or choose the "Save" option from the File menu.) Click on the Main Window to remove any message that may be lingering from previous work we've done (it should have the usual status information in it) and then just select "Check My Disk" from the Custom menu. In a moment, the appropriate message will appear in Frontier's Main Window.

No matter how complex your scripts get and Frontier will let you write some extremely complex and powerful applications this process is pretty much the same. You will probably find yourself doing more iterative definition of variables as you need them, and using multiple scripts so you can get reusable code working for you. Otherwise, the process is almost identical.

Contents Page | Previous Section | Next Chapter -- **A Tour of Desktop Scripts**.

Chapter 3

A Tour of Desktop Scripts

This chapter discusses one of the most powerful aspects of UserTalk: its ability to control the Macintosh desktop via desktop scripts. Through scripts written in UserTalk and then stored in a special format on the desktop, you can give users practically total control over their desktops. In this chapter, we'll take a quick look at some of the desktop scripts that are packaged with Frontier. Then we'll examine two of these scripts closely so that you can see how they work. Next, we'll discuss how you can customize desktop scripts. Finally, we'll explain how to create a desktop script from scratch.

It is neither necessary nor intended that you should understand all of the UserTalk code in this chapter. Our objective is to give you an overview of one important use of Frontier and of its scripting language. But if you are overwhelmed by curiosity, you can look up the UserTalk verbs used in this chapter in the UserTalk Reference Guide.

An Overview of Desktop Scripts

Frontier comes with a number of useful desktop scripts. They are stored in a folder called "Desktop Scripts" on the Frontier distribution disks. Like all good Macintosh applications and goodies, their names tell you a lot about their purpose. For example, the Frontier desktop script called "Find in File" looks through the contents of a file for a string. "Delete Packages" removes AppleLink package files from the folder or volume in which it is stored. You get the idea.

These scripts appear to the user much like other stand-alone Macintosh applications except that they require Frontier to be available. This makes them a cross between a document and an application. Depending on your perspective, you will probably come to think of them as applications or as documents.

There are two things you can do with desktop scripts.

First, you can load them into the Object Database and run them. To do so, you can either double-click on a desktop script icon in the Finder or use the "Open" option from the File menu within Frontier.

In either case, the script becomes part of the Frontier Object Database, where it resides in the table system.deskscripts. Once it exists in the database, you can run it from within Frontier or you can edit it. Be careful, though, if you decide to run a desktop script from inside Frontier. Desktop scripts expect an argument that tells them the file path over which they are to work. In many cases, they use this information to determine their location in the folder structure. If you execute them manually from the Quick Script window, you may have to supply a path name as an argument.

Second, you can edit these scripts in one of two ways. If they are installed in Frontier (as described above), you can edit them like any other Frontier script object. From the Finder, if you hold down the Command key while you double-click on a desktop script in the Finder, Frontier will open the script for editing rather than executing it.

You will have to keep holding down the Command key until the script is fully opened in Frontier. If you release the Command key before Frontier understands that you want to edit the script, it will run it instead. Until Frontier is launched and able to look at what you are doing on the keyboard, it has no way of knowing if you pressed the Command key or not because the Finder application does not inform Frontier of that fact.

Contents Page | Next Section -- **A Closer Look**

HTML reformatting by Steven Noreyko January 1996

Chapter 3: continued

A Closer Look

In this section, we will take a close look at two of the desktop scripts that come with Frontier. For each, we'll examine its purpose and then review the scripts themselves. Our purpose is not to try to teach you a lot about UserTalk at this point; information about the language and how to use it is contained in the later chapters of this manual. If some of the verbs don't seem to make sense to you right now, don't worry about that. We are interested in giving you a chance to sample the flavor of the kinds of operations you can perform with Frontier and some appreciation for how the language works.

Desktop scripts tend to make extensive use of Frontier's verb set that deals with the Finder and the System. They also generally include looping constructs that enable them to operate over several nested levels of folders and files.

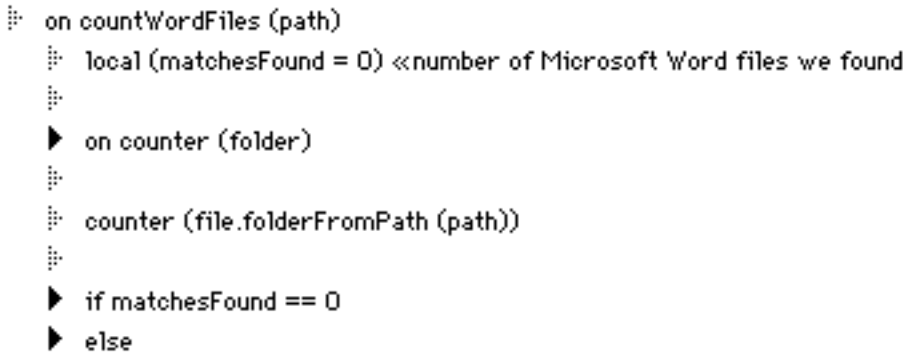
The desktop scripts we'll examine in this section are:

- Count Word Files
- Recently Changed Files

Count Word Files

This desktop script goes through its folders and all of its sub-folders and counts the number of files created by Microsoft Word. It then displays the result.

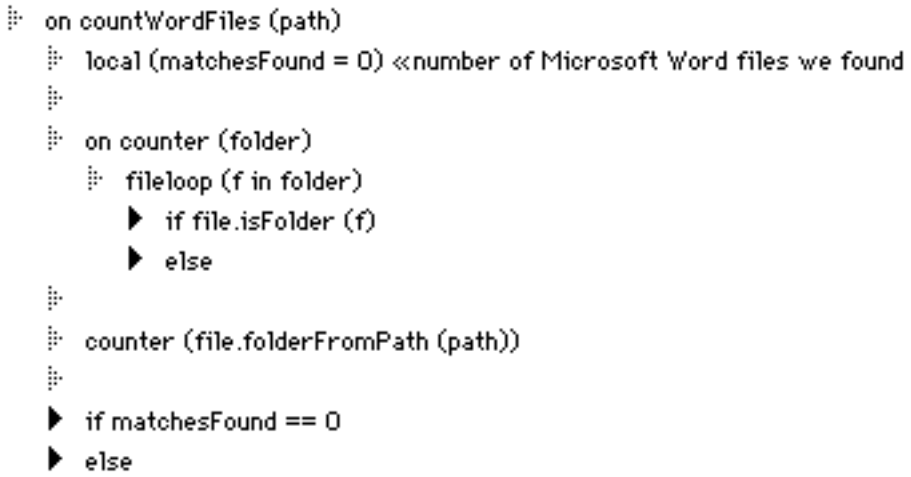
Here is the script for "Count Word Files," as it appears when you open it in Frontier the first time.



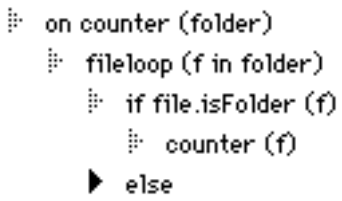
Since a UserTalk script is a form of a Frontier outline and since the Frontier outline, as we saw in Chapter 2, uses solid black item markers to indicate a collapsed heading, you have probably concluded that this script has some collapsed lines. You're right. Using the outline approach to scripting permits you to show the user only as much of a script as you want them to see or as much as you think they'd be interested in. Here, the script is collapsed

enough so that you get an idea of the overall structure of the script. The second line of the script obviously initializes a variable to hold the number of Word files the script locates. Then there is a local script called "counter" embedded in the countWordFiles script. (You can tell it's a local script because it starts with the word "on.") We'll get to that in a moment. Next, we call the local script named counter and pass it the name of the folder in which it is stored (which we find out by using the Frontier verb file.folderFromPath). At the end of the script in other words, after counter has run its course, we see if any files were found and take some action accordingly. As you can see, Frontier gives you the ability to take a very top-level view of a script.

Now let's "drill down" one level deeper into this structure and look at the local script called counter. Here is the full script as we've seen it so far, with that local script open two levels deeper than it was in our last look.



Again, we can see that there are collapsed lines in this local script as well, but we can get a pretty good idea what's going on by looking at what we can see. The first line in the local script called counter sets up a looping construct of some sort. You've almost certainly never heard of a fileloop construct before because it's unique to UserTalk. Without going into detail (you can look it up in the UserTalk Reference Guide if your curiosity just won't let you sleep at this point), this line sets up a loop that will go through every file and folder within the current folder. The next two lines tell us that the routine does one thing if the object it finds is a folder and another thing if it's a file (the only other possibility). So let's take a deeper look at what happens if the object is a folder.



This time we've only shown you the part of the script on which we are focused. There is only one line under the line that tests to see if the object is a folder. It in turn calls the counter local script to iterate over this new-found folder. This is an example of the computer programming concept known as "recursion." A routine is said to be recursive if one or more lines of code within the routine calls the routine itself. This line simply stops when it gets to a folder and tells the counter script to execute on all of the files and (potentially) nested folders in this newfound folder.

What if the object is a file and not a folder? Here's the next level of counter expanded so you can see the next round of detail.

```
on counter (folder)
    fileloop (f in folder)
        if file.isFolder (f)
            counter (f)
        else
            msg (file.fileFromPath (f))
            if file.creator (f) == 'MSWD'
```

If the else portion of the script is executed (which means that we know we're looking at a file and not a folder), the name of the file is displayed in Frontier's Main Window as a kind of progress report. Then the script does another check to see if the file is a Microsoft Word file. It does this by checking the file's creator type, a value assigned by the application developer (in this case, Microsoft) when it writes the program (such as Microsoft Word) that will end up creating document files.

Here's what happens if it is:

```
on counter (folder)
    fileloop (f in folder)
        if file.isFolder (f)
            counter (f)
        else
            msg (file.fileFromPath (f))
            if file.creator (f) == 'MSWD'
                matchesFound++
```

If the file is a Microsoft Word file, we use the increment operator (++) to add 1 to the variable called matchesFound.

This looping process continues until all of the files and folders in the folder from which the desktop script was launched have been examined. Let's look at the last part of the script.

```
if matchesFound == 0
    local (s = file.fileFromPath (file.folderFromPath (path)))
    s = "There are no Word files in the " + s + " folder."
    dialog.alert (s)
```

If the variable matchesFound has a value of zero, then we want to tell the user there weren't any Word files in the folder in which the desktop script is located. If there is at least one Word file in the folder, however, we want to tell the user how many we found.

```
if matchesFound == 0
    local (s = file.fileFromPath (file.folderFromPath (path)))
    s = "There are no Word files in the " + s + " folder."
    dialog.alert (s)
else
    local (s = file.fileFromPath (file.folderFromPath (path)))
    s = "There were " + matchesFound + " Word files in the " + s + " folder."
    dialog.alert (s)
```

The else part of this portion of the desktop script simply displays a dialog box with the number of Microsoft Word files it found.

Here is the script for "Count Word Files," fully expanded to show you every line:

```
on countWordFiles (path)
    local (matchesFound = 0) «number of Microsoft Word files we found
    
    on counter (folder)
        fileloop (f in folder)
            if file.isFolder (f)
                counter (f)
            else
                msg (file.fileFromPath (f))
                if file.creator (f) == 'MSWD'
                    matchesFound++
    
    counter (file.folderFromPath (path))
    
    if matchesFound == 0
        local (s = file.fileFromPath (file.folderFromPath (path)))
        s = "There are no Word files in the " + s + " folder."
        dialog.alert (s)
    else
        local (s = file.fileFromPath (file.folderFromPath (path)))
        s = "There were " + matchesFound + " Word files in the " + s + " folder."
        dialog.alert (s)
```

Now you see what a typical UserTalk script looks like and you have some idea how a desktop script works. We won't go into such detail on the remaining desktop scripts we discuss in this chapter.

HTML reformatting by Steven Noreyko January 1996

Chapter 3: continued

Recently Changed Files

The next desktop script we'll look at is called "Recently Changed Files." As its comment informs us, it creates a folder and then places into that folder aliases of files that have changed since a user-selected date.

Here is what the script looks like in one view, where we've collapsed some of its lines for easier reading:

```
on recentChanges (path)
	« creates a folder full of aliases of files that have changed

	s = "October 1, 1991"
	if not dialog.ask ("Gather files changed since when?", @s)
	marktime = date (s)
	on recentlyChanged (f) «return true if the file matches our search criteria
	on doGather (path)
		local (f)
		fileloop (f in path) «loop over all the files in the path
			if file.isFolder (f) «it's a folder, recurse
			else «it's a file

	folder = file.folderFromPath (path)
	destFolder = folder + "Recently Changed Files:"
	if file.exists (destFolder)
		file.deleteFolder (destFolder)
	if not file.newFolder (destFolder)
		return (false)

	ctgathered = 0
	doGather (folder)

	if ctgathered > 0
	else
	return (true)
```

We first set up a date here it's October 1, 1991 but you could obviously change it and then we use that as the default answer as we ask the user to define which files are to be gathered. We make sure that this value is a date by converting it with the date verb.

The next two sets of lines define local scripts called recentlyChanged and doGather. We'll

come back to them shortly when they are called from the main script.

Before we start gathering aliases of all the recently changed files, we make sure that there's an empty folder whose name is the same as that of the folder in which the script is run, with the words "Recently Changed Files" appended to it. If a folder of this name already exists, we delete it. In either case, we create the folder before proceeding.

Now the script initializes a variable called ctgathered (the "ct" is a mnemonic for "count") to zero and then calls the local script doGather with the name of the folder in which it is running as an argument. Let's examine the doGather local script.

After defining a local variable, it sets up a loop like the one we saw in the last section to loop over all the files in the path. It then takes different actions depending on whether it's a folder or a file, much as we saw with the CountWordFiles script in the last section. Let's open those two sets of lines; the doGather script now looks like this:

```
on doGather (path)
    local (f)
    fileloop (f in path) «loop over all the files in the path
        if file.isFolder (f) «it's a folder, recurse
            if f != destfolder «don't go into the new folder
                dogather (f)
        else «it's a file
            msg (file.fileFromPath (f))
            if recentlyChanged (f) «it matches the local script's criteria
                file.newAlias (f, destfolder + file.fileFromPath (f)) «create the alias
                ctgathered++
```

As you can see, if doGather encounters a folder, it first checks to make sure this isn't the folder we're in the process of creating (or this backup could get very messy!) and then calls itself recursively, just as we saw with CountWordFiles. If the item it's encountered is a file, it displays the name of the file in Frontier's Main Window as a progress indicator, then calls the recentlyChanged local script. That simple script looks like this:

```
on recentlyChanged (f) «return true if the file matches our search criteria
    return (file.modified (f) > marktime)
```

This script consists of one executable line that compares the date the file was last modified against the date the user has indicated we should use as a base date for gathering files. If the file's date of last modification is later than the date the user provided, this routine will return an answer of "true" to doGather. Otherwise, it returns false.

Now you can go back to doGather and see that it checks the return value from recentlyChanged. If it's true, it creates a new alias for the file, places it into the destination folder we're building and then increments the counter that keeps track of how many files we gather by 1.

When all of the files and folders have been examined, the script looks at the value of the variable ctgathered.

```
if ctgathered > 0
    file.openFolder (destfolder)
else
    file.deleteFolder (destfolder)
    local (s = file.fileFromPath (folder))
    s = "There were no recent changes in the “" + s + "” folder."
    dialog.alert (s)
```

If at least one file has been found, it opens the folder so the user can see the results of gathering all of these aliases. Otherwise, it deletes the folder and informs the user that it couldn't find any files to gather.

Customizing Desktop Scripts

The desktop scripts that come with Frontier are robust, but we obviously couldn't anticipate all the things you might want to use them for. Since they're UserTalk scripts, though, you can change them to fit your own purposes. We'll look at two possible changes to these scripts here; realize that these are simply designed to get your creative juices flowing about things you might do with these desktop scripts.

Contents Page | Previous Section | Next Section -- **Counting Other Kinds of Files**

HTML reformatting by Steven Noreyko January 1996

Chapter 3: continued

Counting Other Kinds of Files

The most obvious way you could customize the countWordFiles script would be to look for some other kind of file than Microsoft Word files. You can do this by editing the script in Frontier and making these simple changes:

- Replace "Microsoft Word" in the comment on line 2 with the name of the application whose files you want to find.
- Change 'WBDN' to the four-letter creator code of the application whose documents you wish to find.
- In the two message lines in the last part of the script, change "Word" to the name of the product whose document files you are counting.

Here, for example, is the countWordFiles script, modified to look for Frontier files and renamed countFrontierFiles.

```
on countFrontierFiles (path)
    local (matchesFound = 0) «number of UserLand Frontier files we found

    on counter (folder)
        fileloop (f in folder)
            if file.isFolder (f)
                counter (f)
            else
                msg (file.fileFromPath (f))
                if file.creator (f) == 'LAND'
                    matchesFound++

    counter (file.folderFromPath (path))

    if matchesFound == 0
        local (s = file.fileFromPath (file.folderFromPath (path)))
        s = "There are no Frontier files in the " + s + " folder."
        dialog.alert (s)
    else
        local (s = file.fileFromPath (file.folderFromPath (path)))
        s = "There were " + matchesFound + " Frontier files in the " + s + " folder."
        dialog.alert (s)
```

Notice that when we check for the file creator, we have to ask the script to look for a new

creator type. All files created by Frontier have a creator of 'LAND'.

If you want to get really ambitious, you could use one of the dialog verbs to ask the user for the application whose files are to be counted. There's even a way in UserTalk to let the user point to the application in a standard file dialog box, get that application's four-character creator type, and then use the result to count the files. These, as authors of the college textbooks say, are exercises left to the reader.

Using a Folder's Date as a Starting Point

The "Recently Changed Files" script has the default date entered directly as a value. Programmers call this "hard coding" information and it's generally not a great idea. If you are really disciplined and undertake this process at least once a week, you can easily change the script so that the default date offered by the script when it asks the user how old the files should be is calculated to be one week ago. Just change the line that now reads:

```
s = \"October 1, 1991\"
```

```
s = clock.now () - (7 * 24 * 60 * 60)
```

This statement subtracts from the present day and time the value of seven days times 24 hours per day times 60 minutes per hour times 60 seconds per minute. It produces a time exactly 168 hours before the present time.

You could use other approaches to the problem of the hard-coded date. For example, you could check to see if the folder called "Recently Changed Files" exists and, if it does, use the file.modified verb to get its last modified date. This would enable you to gather files that have changed only since the last time you gathered changed files.

Creating Desktop Scripts

Desktop scripts are not much different from other UserTalk scripts. Most of them deal with operations that make sense at the Finder level, involving things like files, folders, and system issues. But that is not necessary. Any UserTalk script that you'd like a user to be able to launch by double-clicking it in the Finder can be made into a desktop script.

Because it is designed to be called from a place other than Frontier's Quick Script window (namely, from the scripts that facilitate the use of desktop scripts within Frontier), a desktop script must use the on keyword approach to its format. (This subject is covered in greater detail in Chapter 6.) It must also take a single argument, which Frontier will interpret as the path of the folder in which it is stored (or the volume on which it is located if it is not placed into a folder).

To make a UserTalk script into a desktop script, all you have to do is choose "Export a Desktop Script..." from the UserLand menu in Frontier. You can also use this command's keyboard equivalent, Command-3. You'll be asked (Figure 3-1) which script you want to export as a desktop script. The name of the selected object will be offered as a default response.

Figure 3-1. Dialog for Exporting a Script to the Desktop

Once you indicate the name of the object you wish to export, either by accepting Frontier's proffered response or by typing in your own table address, you will be asked to name and position the file in which to store this script. This interaction uses the standard Macintosh file dialog.

Before you give a desktop script to another user to work with, we recommend that you test it on at least one system other than your own. This will enable you to discover common mistakes such as including a reference to a disk volume name in your script, which is fine as long as you are only running the script on your system. Other users, however, will encounter errors in such situations.

That's all there is to it. If the folder or volume into which you place the desktop script is open at the time, you may have to close it and then re-open it for the desktop script to be able to be launched with a double-click; this is a characteristic of the Macintosh system with files that are placed into open folders or volumes from external sources such as Frontier or a telecommunications program.

Contents Page | Previous Section | Next Chapter -- **The Frontier Environment**

HTML reformatting by Steven Noreyko January 1996

Chapter 4

The Frontier Environment

This chapter discusses all aspects of the Frontier environment with the exception of the Object Database (which is covered in Chapter 5) and scripting (which is the subject of the remainder of this manual). Specifically, this chapter will describe Frontier's menus, how to navigate in and use Frontier's various types of windows, and how to use the script-related windows.

You should also recall from our discussion in Chapter 2 that you can use the Main Window's buttons to access some of the elements of the Frontier environment described in this chapter.

Frontier's Menus

Because of its extensible nature, Frontier has more menus than appear when you open it. In addition to the standard Apple, File, and Edit menus, you will always see four other menus in Frontier: UserLand, Custom, Suites, and Window. Depending on what type of window is frontmost in Frontier, you will also see one of the following menus: Script, Table, or WP. Beyond this, the use of various suites of scripts included with Frontier can result in the Map and Glue menus appearing, among others.

As you customize your Frontier environment, either with UserTalk scripts and suites of your own creation or with those you obtain from others, more menus may well become available.

In addition, you can customize all of Frontier's menus except the four permanent ones (Apple, File, Edit, and Window).

In this section, then, we will document each of these menus as they are shipped from UserLand Software. If your menus differ somewhat, it's because someone has already been about the business of customizing them.

Scripts and the Special Menus

All of the menu items in Frontier with the exception of those in the four permanent menus invoke UserTalk scripts to carry out their operations. If you are curious about what a menu item's script does, you can find out in one of two ways.

First, you can hold down the Option key as you pull down the menu. Select any menu item and then release the mouse and the Option key. You will be placed into Frontier's menubar editor. Select the menu item whose script you want to see and click on the Script button in that editor.

NOTE

You can save yourself one step here by choosing the menu item you wish to work with as you pull down the menu rather than finding and selecting it after the menubar editor is open.

Second, if you know the name of the script invoked, you can locate it with the Jump option from the UserLand menu (or use its Command-J keyboard equivalent) and enter the name of the script in the resulting dialog (see Figure 41). Frontier will open the script for you.

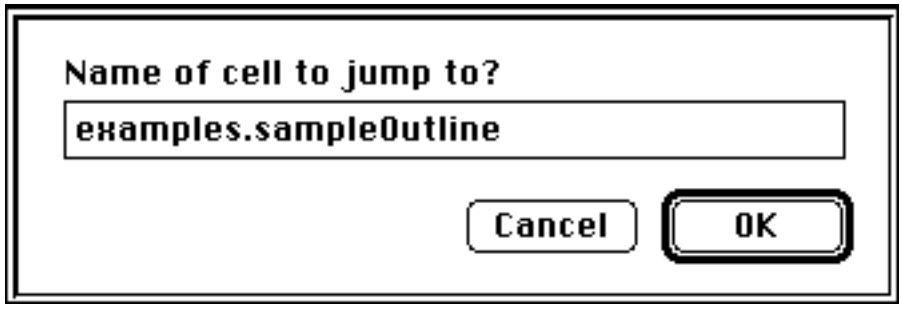


Figure 41. Jump Dialog

A Word About Terminology

Before we start our description of each menu in Frontier, we should be clear about some of the terms we'll be using in this chapter.

The menubar is the entire top line of the display of your main (or only) Macintosh monitor. It is the place where all of the menus show up.

A menu is an individual label entry on the menubar. When we speak of the File menu, we mean the word File on the menubar and all of the items on that menu when it is opened by clicking on its label with the mouse.

A menu item is an individual entry in a menu. We will speak of the Save menu item on the File menu, for example. When we discuss selecting a menu-based action, we will often use words like choose and select with respect to menu items, which we will also sometimes refer to as choices or options. We use the term item to mean menu item as well.

Apple Menu

The Apple menu is standard on all Macintoshes. The only item on this entry that is unique to Frontier is the About UserLand Frontier... option that appears first in the menu. When you select this item, you'll be shown the splash screen message that appears when you start Frontier each time. This message will disappear when you click the mouse anywhere in the Frontier environment or press any key on the keyboard.

The rest of the items on the Apple menu appear there because they are stored in the Apple Menu Items folder of your System Folder if you are using a Macintosh under System 7, or because they are defined as desk accessories if you are using System 6.

Contents Page | Next Section -- **File Menu**

HTML reformatting by Steven Noreyko January 1996

Chapter 4

File Menu

The File menu (Figure 42) contains standard Macintosh File menu entries. There are two noteworthy things about your use of this menu, however.

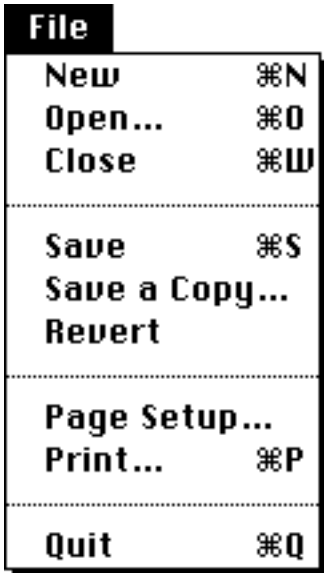


Figure 42. File Menu

If you choose the Open... option from this menu, Frontier's action will depend on what you choose to open. If you open a new version of the root file, Frontier does not close the one that you had previously opened. If you open a desktop script, Frontier will load it, store it in the proper place and, if possible, execute it. If, on the other hand, you open an exported Frontier object, Frontier will ask you where in the Object Database to store it and will then store it in the indicated location.

The Print... option on the File menu prints the contents of the frontmost window. If that window is a table, it prints only the table itself, not the contents of each object stored in the table. If you want to create a printout of a table and all its contents, you will have to write a UserTalk script.

The Revert option will revert all changes to the current version of the root file. Before doing so, Frontier will confirm your intention.

Edit Menu

The Edit menu is almost entirely identical to other Macintosh Edit menus you've seen. It is shown in Figure 43.

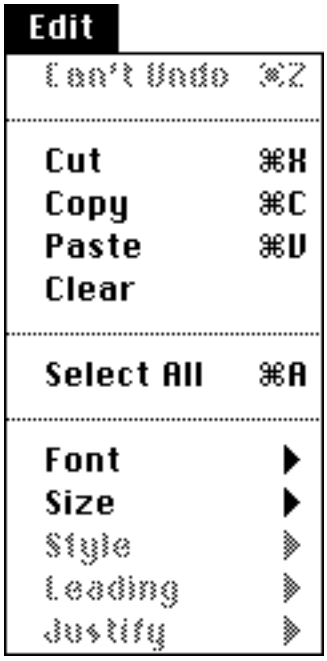


Figure 43. Edit Menu

You may not be accustomed to seeing the font and styling options on the Edit menu. You'll notice that Frontier dims the last three choices unless you are editing a word processing text document. The first menu item on the Edit menu is context-sensitive. It changes from Can't Undo to Undo Move, for example, if you have just moved a headline in an outline document.

The Select All option on this menu selects all of the text in a word processing text document. In a script or outline, it selects all of the text in the current heading. In a table, it selects the entire name of the current object.

HTML reformatting by Steven Noreyko January 1996

Chapter 4: continued

UserLand Menu

The UserLand menu is the first of several customizable menus in the Frontier environment. It is shown in Figure 44. It includes a number of menu options you'll find essential as you write UserTalk scripts.

Figure 44. UserLand Menu

The first selection displays a word processing text document that explains the menu and some of its more important items.

The next selection creates a backup copy of Frontier.Root in the folder from which you launched Frontier. The backups are serialized, so as you work, you leave a trail behind you. Files are named Frontier.Root.361, Frontier.Root.362 and so on. Check out the backups folder in the folder you run Frontier from to see the trail of files. Backing up is important, and this command makes it simple to do. Just press Command-B to be sure your changes are safe. This menu option calls the script backups.backuproot, which you can examine with Command-J as described earlier.

The third selection opens Frontier's menubar editor (see Figure 45). This is the tool you can use for customizing any of Frontier's special menus, including adding your own menu items, renaming or reordering those that are provided by Frontier, deleting menu items, or changing

the behavior of menu items.

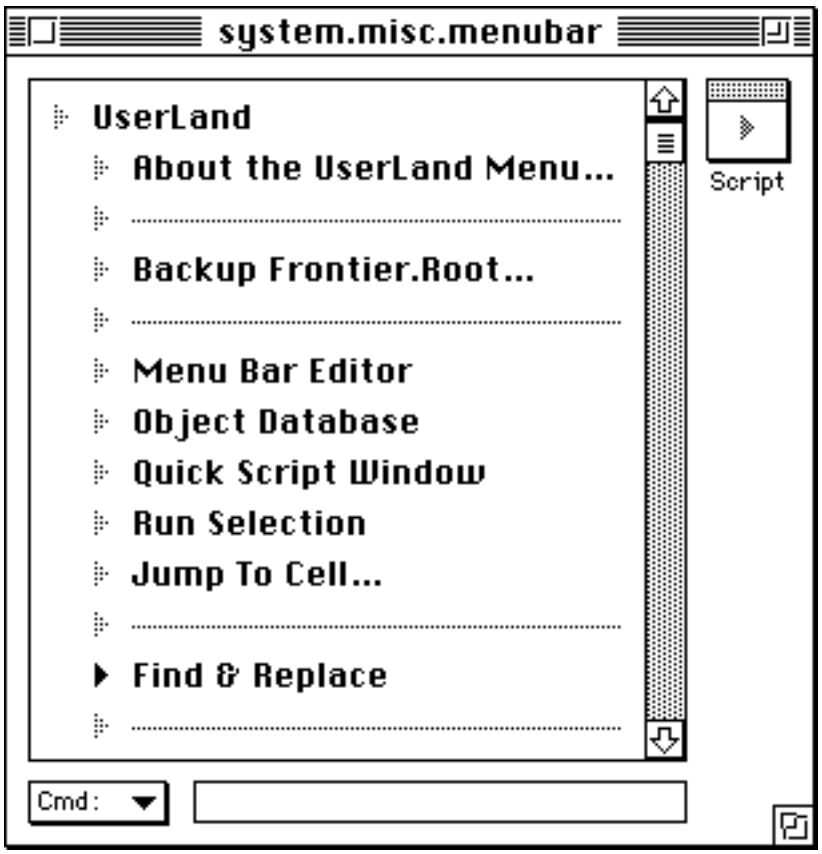


Figure 45. Frontier's Menubar Editor

The menubar editor shows that menus are arranged as outlines in Frontier, with the item markers (triangle-shaped objects) on the left side of the window indicating whether a menu item has sub-headings (if it is black) or not (if it is gray). If a menu item has sub-headings, that means it appears in the menu as a hierarchical menu with a triangle to its right side. Selecting a hierarchical menu item results in the display of additional options to the right of the item selected (see Figure 46).

Figure 46. Sample Hierarchical Menu

Figure 47 shows the same hierarchical menu in the menubar editor. Notice that the three sub-menus from Figure 46 are sub-headings in the outline of Figure 47.

Figure 47. Partial Menubar Editor Showing Hierarchical Menu

In the Frontier menubar editor, you can not only edit the names of menu items and the scripts attached to them, you can also assign or change the Command key associated with a menu item. Just click on the popup menu in the lower left corner of the window (see Figure 48) and choose Set Command Key... to change the current setting or add a Command-key equivalent to a menu item that does not presently have one assigned.

Figure 48. Popup for Command Key Assignment in Menubar Editor

If you choose the first option, Frontier asks (see Figure 49) which Command key you wish to assign to the selected menu item. To remove a Command-key assignment, press Backspace in the dialog.

Figure 49. Command-Key Input Dialog

Frontier looks for Command-key equivalents from right to left on the menubar and from top to bottom within a menu. So if you assign the same Command key twice on a menu, Frontier will only find the first occurrence of the Command key each time it is used. Avoid assigning duplicate Command keys in the same menu and only use duplicate keys in two or more menus when you are sure what the effect on the user will be.

Object Database, or Command-T, (for Table) opens the top level window of Frontier's Object Database (see Figure 410). This is a hierarchy of tables, numbers, strings, outlines, scripts and lots of other types of data, that combined make each installation of Frontier unique. These values can be created by UserTalk scripts, by IAC-aware applications communicating with Frontier, or by the user interactively browsing the object database, using this command. Chapter 5 is devoted to a discussion of the Object Database.

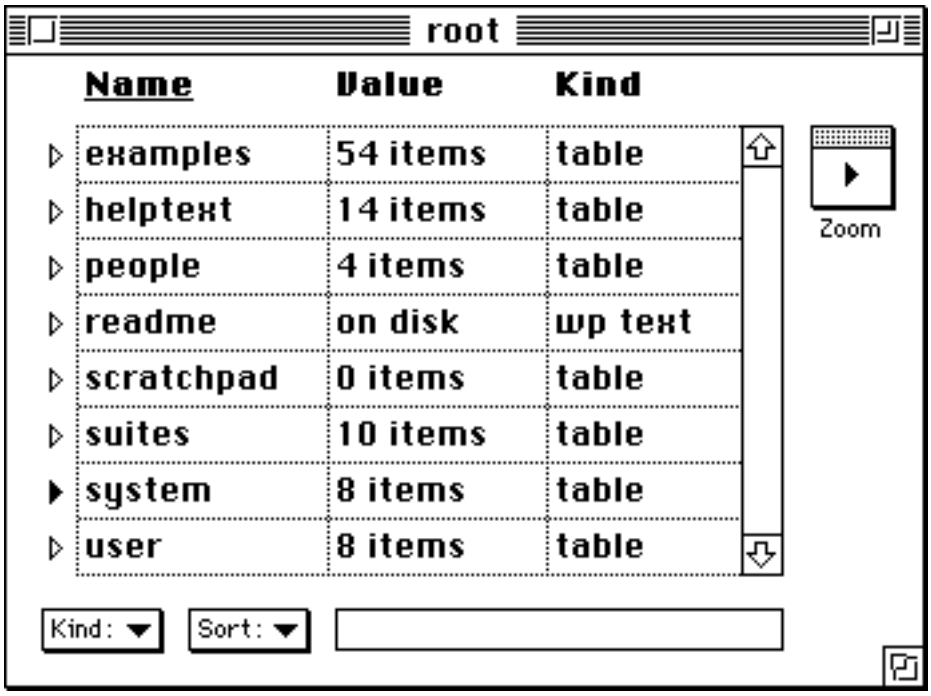


Figure 410. Top-Level Window of Frontier Object Database

Contents Page | Previous Section | Next Section -- **Userland Menu, continued**

HTML reformatting by Steven Noreyko January 1996

Chapter 4: continued

UserLand Menu, continued

Quick Script Window, or Command-semicolon, opens the Frontier Quick Script window (see Figure 411). This window allows you to type in one-line scripts that run immediately. Type in the script, and press Enter or click on the window's Run button. Whatever value is generated by the script (all scripts generate values) is displayed in the message area at the bottom of the window.

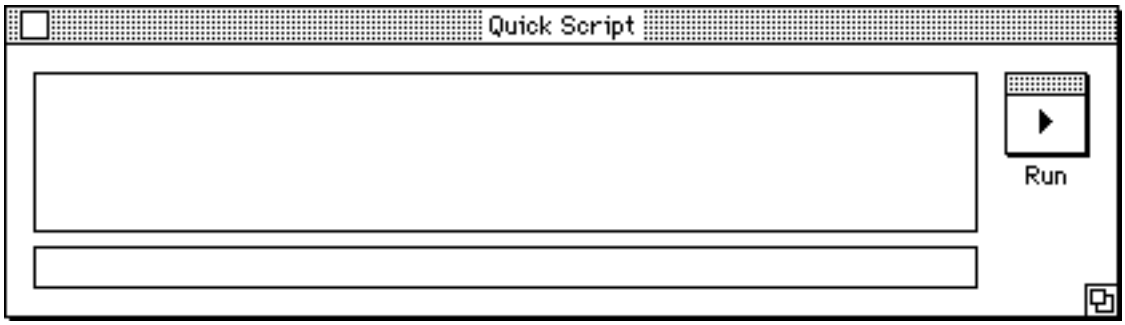


Figure 411. The Quick Script Window in Frontier

Run Selection, or Command-slash, attempts to run the currently selected text in an outline or word processing text document. In outline windows, this option runs the line and places the generated value as the first subhead of the line. In word processing windows the selected text is run, and the return value is discarded. In menubar windows, the script linked to the item is run. In table windows, the selected string or script is run if possible.

Jump to Cell..., or Command-J, provides a way to move directly to a specific cell location in the Object Database. Choosing this option displays a dialog like the one shown in Figure 412. Just type in the name of the table address you want to look at, press Return (or click the OK button) and Frontier will open it for you.

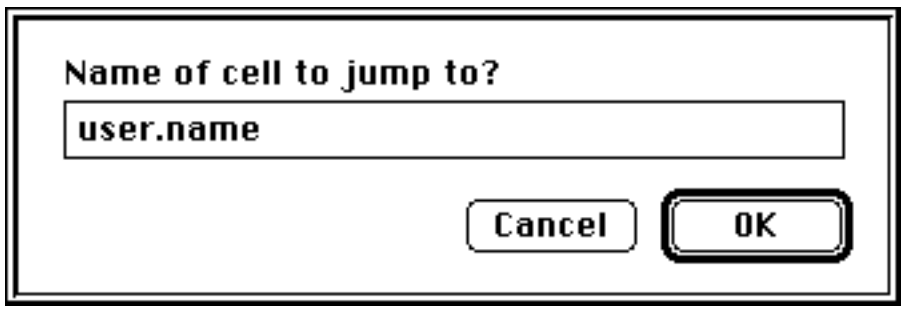


Figure 412. Dialog for Jump to Cell... Menu Option

The Find & Replace menu item is a hierarchical menu whose sub-menus we've seen before.

All of this item's sub-menus have Command-key equivalents and you'll probably find yourself using the find-replace capability in Frontier much more often with those keyboard shortcuts than with the menu options.

Command-F invokes the Find & Replace Dialog option, which in turn displays a dialog like the one shown in Figure 413.

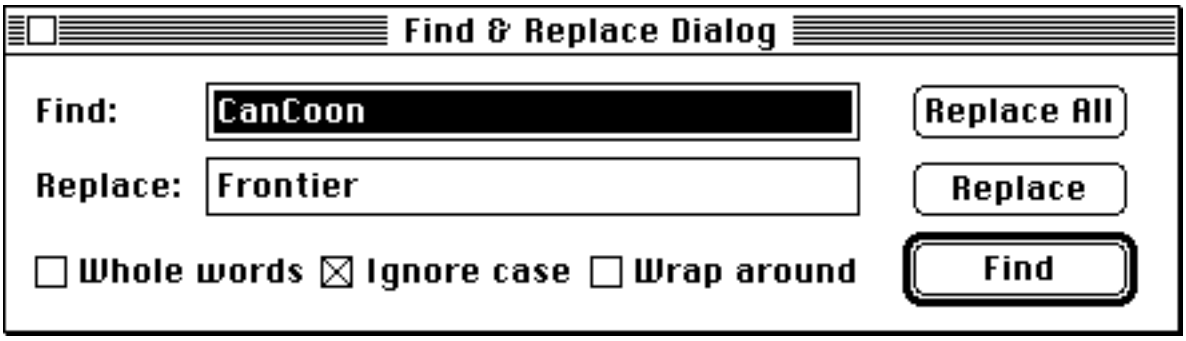


Figure 413. Frontier Find & Replace Dialog

Command-G repeats the last find operation from the current cursor position, looking for the next occurrence of the string for which you are searching. Command-H carries out the replacement operation called for in the Find & Replace dialog and then repeats the last find operation from the current cursor position.

The Agents option is another hierarchical menu. A Frontier agent is a UserTalk script that runs as a background process, repeating its operations on a scheduled basis. The basic concept and use of agents are explained in Chapter 6. This menu item's choices are shown in Figure 414.

Figure 414. Options on Agents Hierarchical Menu

The first option, Open Agents Table, displays the table system.agents (see Figure 415). This is where all agent scripts in Frontier are stored. It is like any other table; you can open its contents and edit them, add to the table, and so forth.

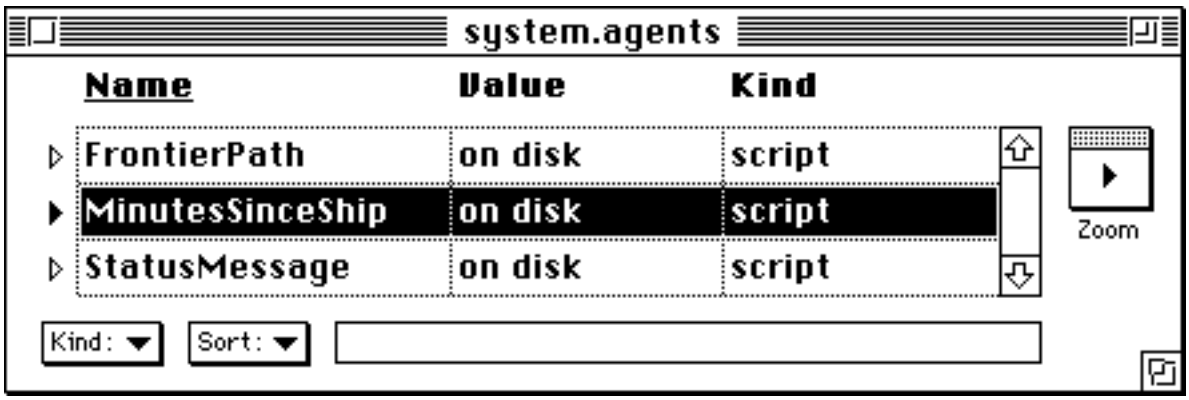


Figure 415. Agents Table Open for Editing

The other two options on the Agents menu item's popup menu turn agents on and off at the user's direction. Turning agents off can make scripts execute more quickly. But all agent scripts are either running or not as a group. You cannot turn one agent off and leave others on other than by cutting an agent script from the agents table and pasting it somewhere else for later retrieval.

NOTE

If agents are turned off, Frontier's automatic changing of menus based on which window is frontmost (which is discussed later in this chapter) will not work. That behavior depends on an agent script.

Common Styles has a hierarchical set of options shown in Figure 416. These are four of the more common and useful type font and size combinations we've found in experimenting with Frontier.

Figure 416. Common Styles Menu from UserLand Menu

We encourage you to change these entries to suit your own style and preferences for font and size combinations. To do so, just follow the instructions at the beginning of this chapter to open one of these scripts. They all look the same except for their specific font and size contents. Here's the one for 9-point Geneva, for example:

```
editMenu.setFont ("Geneva")
editMenu.setFontSize (9)
```

By editing either or both of these lines of script to call for a different type and/or size, you can customize this menu easily. Don't forget to rename the menu item so you'll remember what it does. You could also add your own styles to this menu by copying a script from one of the existing entries and then editing it after attaching it to a newly created entry in the menu.

The Export menu item produces the hierarchical menu shown in Figure 417. As you can see, this menu is divided into three functional groups. The first displays a word processing text object that explains this sub-menu's use. The second is used to export any Frontier object so that it can be transferred to another root file. The last three involve desktop scripts (see Chapter 3).

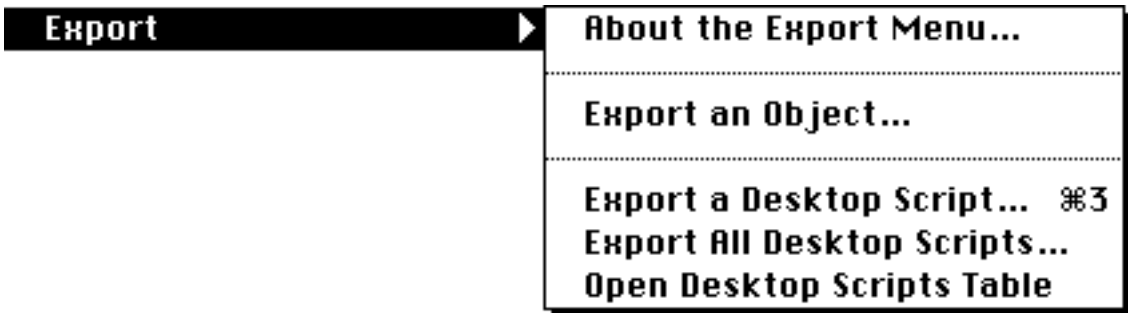


Figure 417. Export Hierarchical Menu from UserLand Menu

Choosing Export an Object... produces a dialog box like the one shown in Figure 418. Just type in the name of the object you wish to export, then provide it with a file name at the next prompt and Frontier will store the object in an external file that can be transferred to other Frontier users. (The use of exporting and importing operations to modify and save portions of the Object Database is discussed in Chapter 5.)

Figure 418. Dialog for Exporting Object in Frontier

A similar process ensues if you choose Export a Desktop Script... from this menu (or invoke the option with the Command-3 keyboard shortcut). This process is described in greater detail in Chapter 3.

Export All Desktop Scripts... produces a similar result, except that it will individually export every script in the table system.deskscripts into a folder called Desktop Scripts in Frontier's current directory.

The last item in the Export menu item's hierarchical menu is Open Desktop Scripts Table. It opens the table called system.deskscripts for editing (see Figure 419).

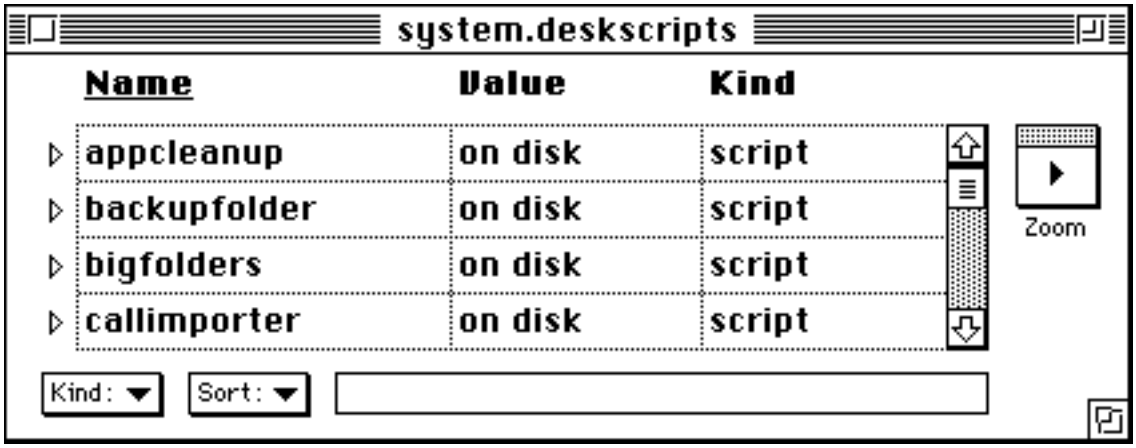


Figure 419. Desktop Scripts Table

The On-Line Docs menu item in the UserLand menu is the last entry. Like the four preceding it, it produces a hierarchical menu, which is shown in Figure 420.

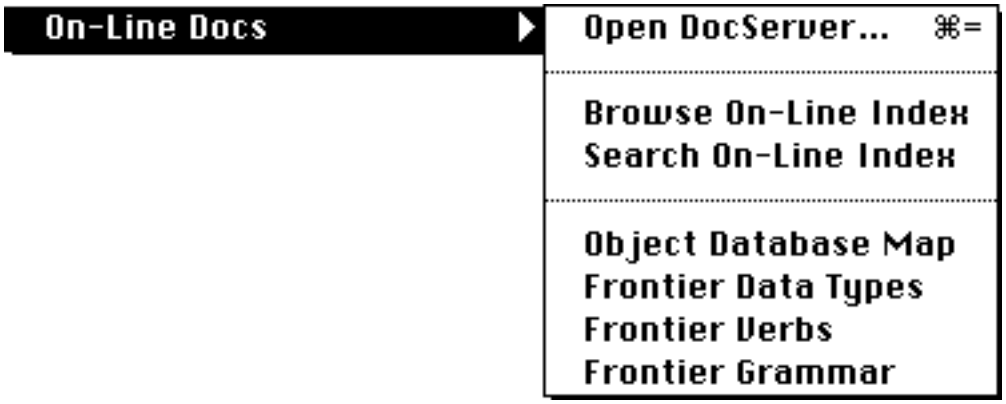


Figure 420. On-Line Docs Hierarchical Menu from UserLand Menu

The various items you can access from this menu are explained in detail in Chapter 7 when we discuss the UserTalk language and how to obtain on-line assistance with scripting. Since their proper use entails an understanding of the language on some level, we will not discuss them here.

Contents Page | Previous Section | Next Section -- **Custom Menu**

HTML reformatting by Steven Noreyko January 1996

Chapter 4: continued

Custom Menu

Figure 421 shows the Custom Menu as it is configured when Frontier leaves the UserLand Software shipping department. This menu is the one you are most likely to want to make your own, so its appearance is almost certain to vary over time.

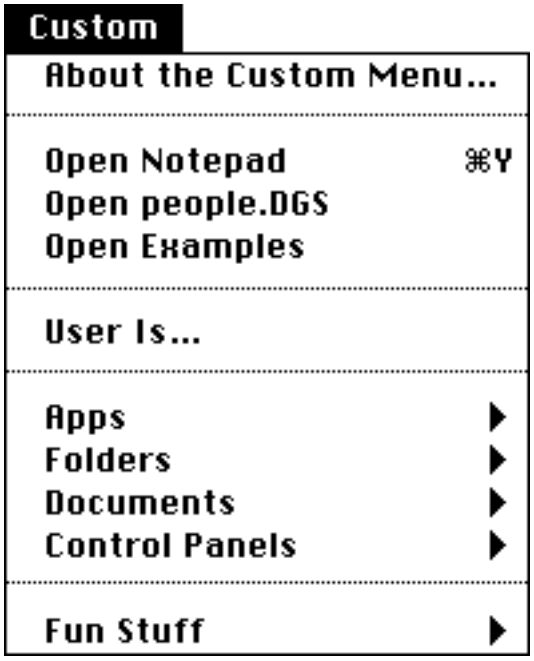


Figure 421. Frontier's Custom Menu

As with most of the Frontier menus, the first item in this menu opens a word processing text document that describes this menu and its use. The next three items open specific objects: Notepad (an outline which is automatically part of your people table and which you may find useful for storing lists and reminders), your people table (where you can store a host of personal objects), and the table root.examples (which is used extensively in the UserTalk Reference Guide). The second of these menu items will have the initials of the current user appended to the people label.

With the User Is... option, you can change the identity of the person using this copy of Frontier. While only one user at a time can use any copy of Frontier, you can switch users when necessary. If you choose this option, you'll be shown a dialog like the one in Figure 422. This is identical to the one you see when you first install Frontier. By changing the user's name and initials or just the user's initials you can allow a different user with a separate people table to use this copy of the Object Database.

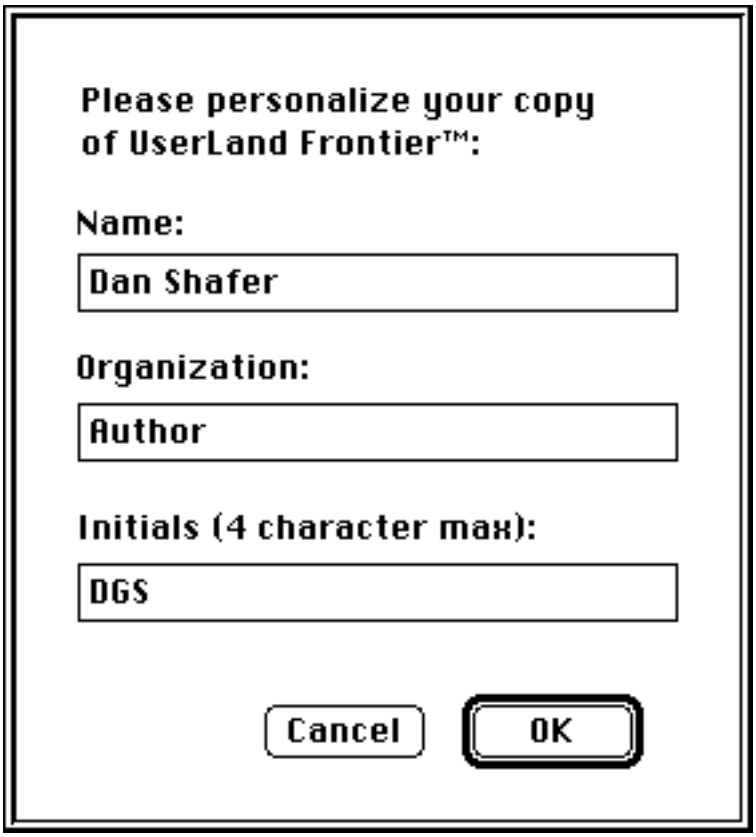


Figure 422. Personalization Dialog for Changing Users in Frontier

The next four items in the Custom menu are quite similar to each other. They are designed as places for you to be able to add your favorite applications to launch, folders to open, documents to edit in other applications, and control panels to open. You can even add other types of things you might want to launch if you have others that aren't covered by these four categories.

As they come to you from UserLand, these menus have hierarchical sub-menus that open, among others, the DocServer and BarChart applications that are part of the Frontier product line that you receive, the Frontier and System folders, three About... documents that also come with Frontier, and two Control Panels: Startup and Monitors.

Figure 423 shows what the Documents option looks like.

Figure 423. Sample Popup From Custom Menu

Contents Page | Previous Section | Next Section -- **Suites Menu**

HTML reformatting by Steven Noreyko January 1996

Chapter 4: continued

Suites Menu

The Suites menu (see Figure 424) provides a launching pad for a number of suites included with Frontier. Suites are collections of related UserTalk scripts attached to one or more private menus. The subject is discussed in greater detail in Chapter 6.

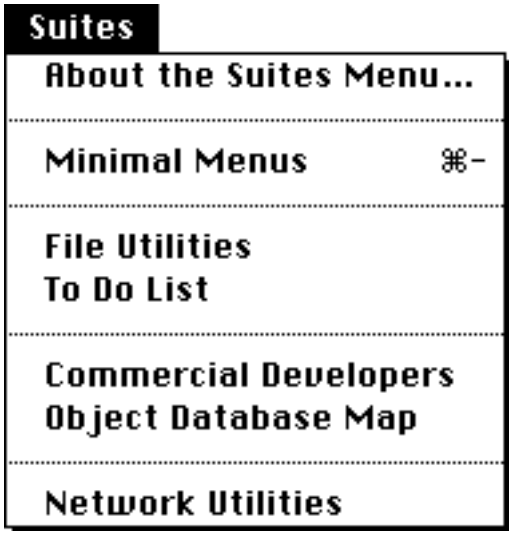


Figure 424. Suites Menu

The first item on this menu, About the Suites Menu... is similar to the other About... menu items we've seen. It explains this menu and its use in a word processing text document.

Minimal Menus, or Command-hyphen, is quite useful. When you've finished using a suite, you can use this menu option to remove its menu from the menubar.

The rest of the items on this menu generate their own menus, which are discussed later in this chapter.

Window Menu

The Window menu is in many ways more closely related to the three permanent menus in Frontier than it is to the others we've discussed. Its contents are not scriptable, but they are dynamic and Frontier-specific. At any point in time, the Window menu contains a list of all of the windows that are open in Frontier. Figure 425 shows you a typical example of this menu in use.

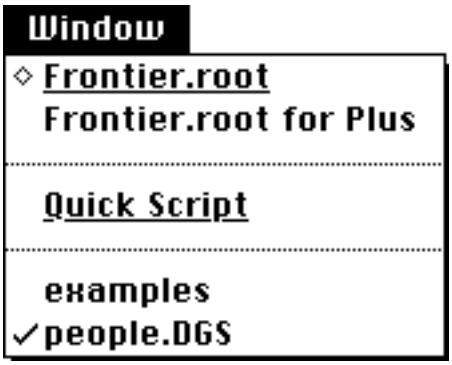


Figure 425. Typical Window Menu

Each of the symbols and styles in this menu means something. A diamond character is used to indicate the current root file in use. (You may have more than one open, as shown in Figure 425.) An underlined window name means that object's contents have changed since they were last saved. The checkmark identifies the frontmost window.

Contents Page | Previous Section | Next Section -- **Outline Menu**

HTML reformatting by Steven Noreyko January 1996

Chapter 4: continued

Outline Menu

Whenever an outline object's window is frontmost in Frontier, a menu labeled Outline appears to the left of the Window menu. The Outline menu is shown in Figure 426.

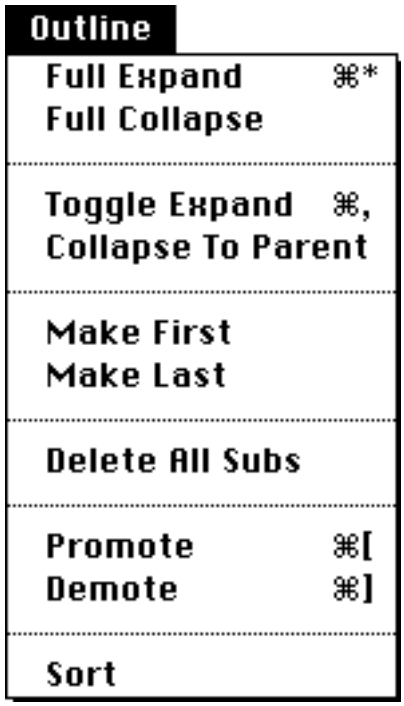


Figure 426. Outline Menu

NOTE

You may wonder how Frontier changes its menus dynamically as various windows become frontmost. The secret lies in a script called suites.modes.monitor. This script is called by system.agents.statusMessage. You can examine this script and see how Frontier handles the change of menus when you change object types.

Full Expand and Full Collapse change your view of an outline's contents. The former opens all levels of sub-headings in the document so all lines are visible. The latter collapses headings so that only summits (of which there may be more than one in a Frontier outline) are visible.

Toggle Expand, or Command-comma, switches between expanded and collapsed views on the selected heading. Collapse to Parent hides the currently selected heading and any sub-headings it has, collapsing them under the next higher level heading in the outline. Figure 427 shows a sample outline fully expanded.

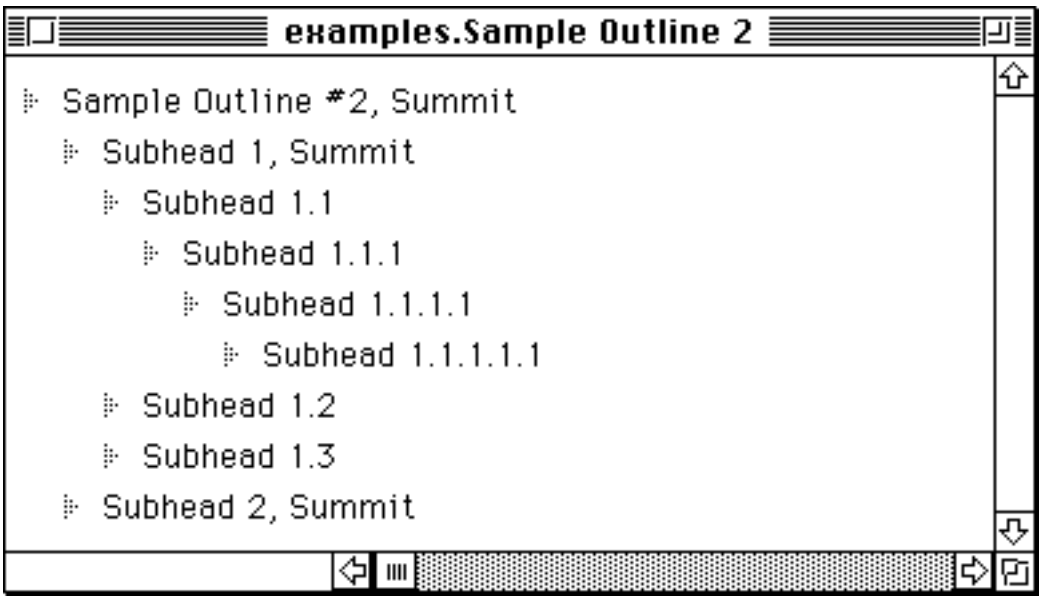


Figure 427. Sample Outline, Fully Expanded

In Figure 428, we positioned the bar cursor on Subhead 1, Summit, and chose Toggle Expand from the Outline menu. Notice that the selected heading's sub-headings are all collapsed under it. In Figure 429, we chose Toggle Expand again. Notice that only the next level headings are expanded under Subhead 1, Summit. That's because this is not a full expand but merely a single-level expand.

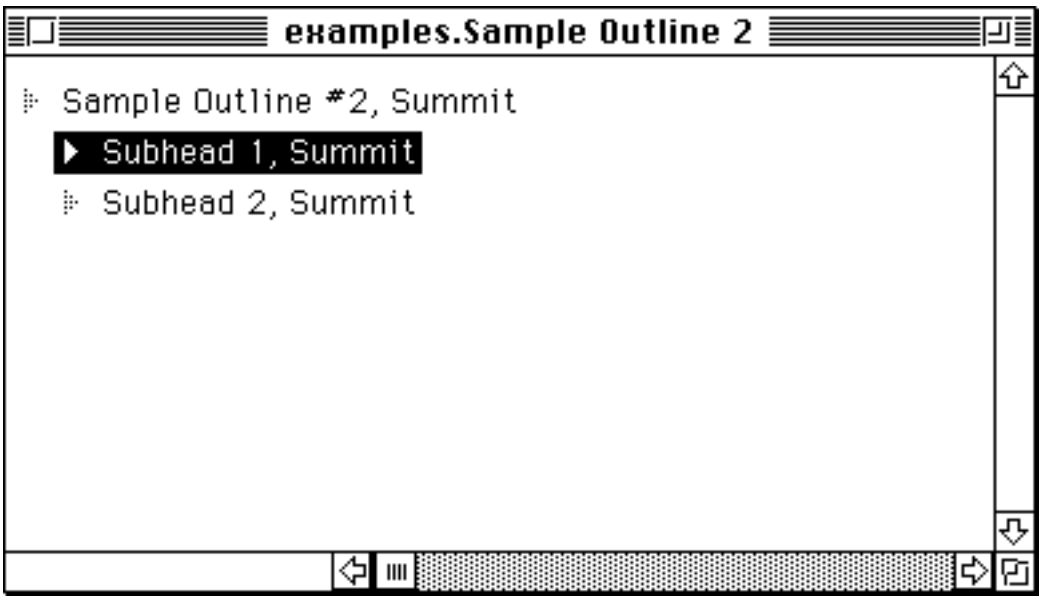


Figure 428. Sample Outline With Sub-Heading Collapsed

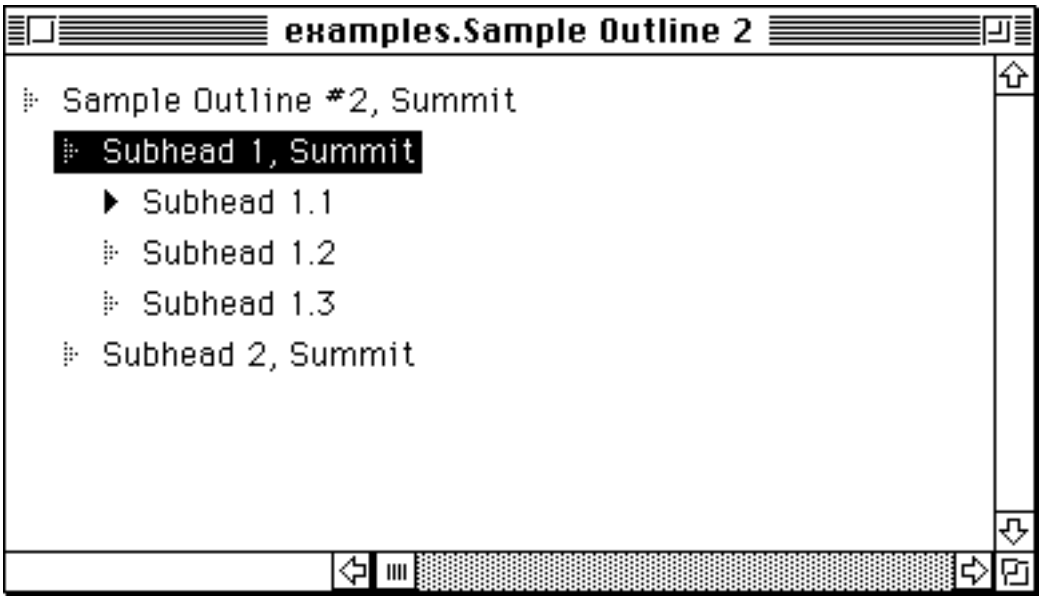


Figure 429. Toggle Expand Opens Next Level of Headings

Make First and Make Last rearrange headings at the same level that are children of the same parent. By selecting a heading at a particular level where there are multiple headings sharing a parent, you can use these two menu options to move individual headings and their sub-headings around within their portion of the outline. In Figure 430, for example, we selected Subhead 1, Summit, which was the first sub-heading under the summit, and chose Make Last from the Outline menu to relocate it.

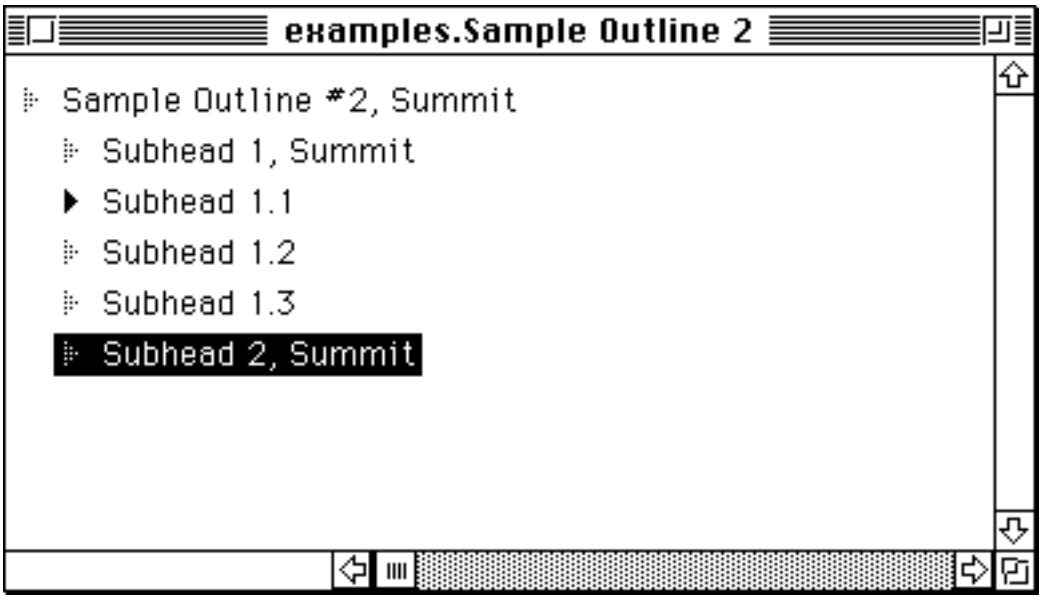


Figure 430. Example of Use of Make Last in Outline

Delete All Subs deletes all sub-headings that are subordinate to the selected heading.

Promote and Demote increase and decrease, respectively, the levels of the headings immediately subordinate to the selected heading. Figure 431 shows the sample outline we are working with (shown in its original form in Figure 427) after we positioned the bar cursor on the headline Subhead 1, Summit and then chose Promote. Note that the three

sub-headings that were formerly subordinate to the selected heading are now at the same level as their former parent. Demoting headings has the opposite effect.

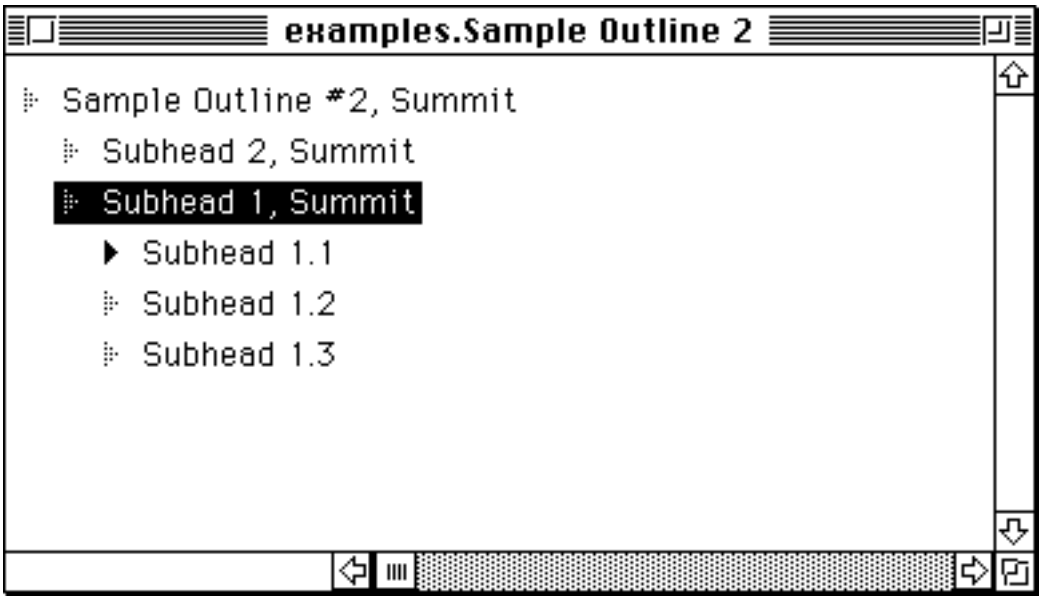


Figure 431. Demonstration of Promote in Use

Sort arranges the selected heading and all of its siblings in alphabetical order. The order of sub-headings under siblings is not affected.

Contents Page | Previous Section | Next Section -- **Script Menu**

HTML reformatting by Steven Noreyko January 1996

Chapter 4: continued

Script Menu

When a script-editing window is the frontmost window in Frontier, a menu labeled Script appears to the left of the Window menu. The Script menu is shown in Figure 432.

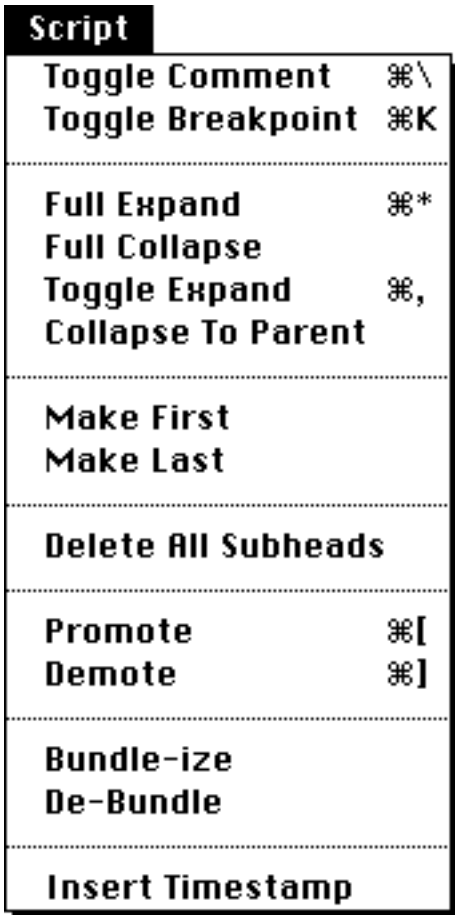


Figure 432. Script Menu

This menu, as you can see, has a large number of options. Some of them are related to the fact that Frontier uses an outlining metaphor for programming.

Toggle Comment, or Command-backslash, converts a line in a scripting window from a comment to an executable line of UserTalk code and vice versa. A comment is indicated by a chevron () while an executable line of code is marked with an outline item marker. Any heading(s) nested under a line that you convert to a comment are also converted to comments by this menu option. Similarly, toggling a comment with sub-headings back to an executable line of code converts its sub-headings to executable lines as well. The exception to this rule is that a sub-heading you explicitly enter as a comment will never be toggled to an

executable line of code by the toggling of a higher-level heading. You'll have to convert such lines individually between comments and executable lines if the need arises.

Toggle Breakpoint, or Command-K, sets and removes breakpoints from UserTalk scripts. A breakpoint is indicated by a hand character (see Figure 433). When UserTalk encounters a breakpoint, it halts execution. The use of breakpoints as debugging tools is discussed in Chapter 6.

Figure 433. Breakpoint Character

Full Expand, Full Collapse, Toggle Expand, and Collapse to Parent all have the same effect when applied to scripts as they do when applied to an outline. The same is true for Make First and Make Last. Delete All Subs, Promote, and Demote behave identically to their Outline menu counterparts as well.

The next two items Bundle-ize and De-Bundle relate to the concept of bundling groups of lines in your UserTalk scripts. The first item creates a heading with the keyword Bundle and demotes all of the headings beneath it so that they become a single group of lines that you can then collapse. The second item is meant to be used when you are positioned on a line with the word Bundle as its first entry or on any line contained in a bundle. It undoes the effects of a Bundle-ize option. We have more to say about bundling in Chapter 6.

Insert Timestamp is a way for you to keep track of versions of your UserTalk scripts. When it is selected, it inserts a comment line into the script, on the line immediately following the currently selected line. This comment displays the date and time and the user's initials (see Figure 434).

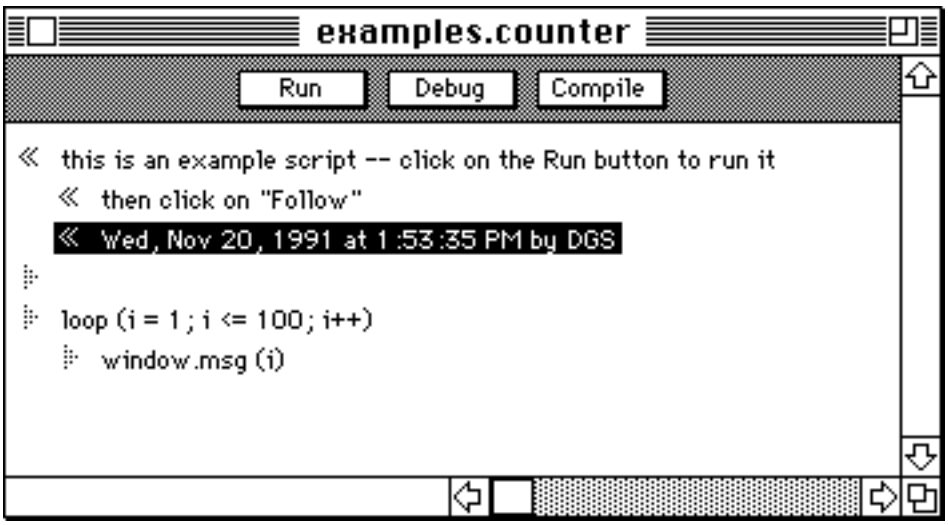


Figure 434. Sample Timestamp

Table Menu

Whenever a table object's editing window is frontmost, Frontier displays a menu labeled Table to the left of the Window menu. The Table menu is shown in Figure 435.

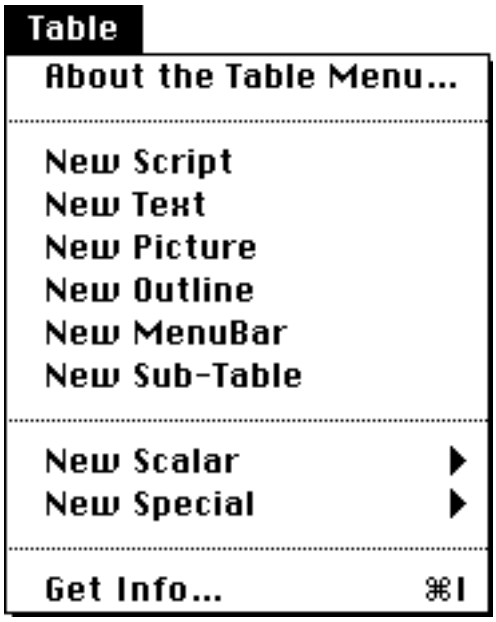


Figure 435. Table Menu

As with the other special Frontier menus, the first option displays a word processing text object that explains the menu and its use.

The next eight menu items each allow you to create a new entry in the current table. They all work fundamentally the same: they ask you for the name of the new object, then they add it to the table. Once you add a new scalar, you can edit its value, which defaults to zero or empty, by selecting its value and editing it .

Scalars are:

- Boolean
- Character
- Number
- Float
- Date
- Direction
- String

Special types of data are:

- String4
- Fixed
- Point
- Rect
- Pattern
- Color
- File Spec
- Alias
- Address
- Binary

Most of the special types of data will be of little concern to you until you've had some

experience with UserTalk. They are also scalars, but are separated here because of their more limited usage than the more common scalars in the first list. All of the datatypes are explained in the UserTalk Reference Guide.

Contents Page | Previous Section | Next Section -- **WP Menu**

HTML reformatting by Steven Noreyko January 1996

Chapter 4: continued

WP Menu

When a word processing text object's editing window is frontmost in Frontier, a menu labeled WP appears to the left of the Window menu. The WP menu is shown in Figure 436.

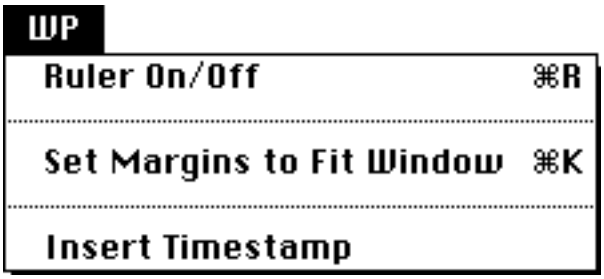


Figure 436. WP Menu

Ruler On/Off, or Command-R, toggles the ruler in the word processing text window on and off.

The second option, Set Margins to Fit Window, or Command-K, changes the margins of the document in the target word processing text window so that all of the text is visible.

Insert Timestamp works the same as the same menu item on the Script menu, discussed earlier.

Map Menu

The Map menu (Figure 437) appears when you choose the Object Database Map from the Suites menu. At the same time as the Map menu appears, so does an Outline menu since the map is presented in the form of an outline.

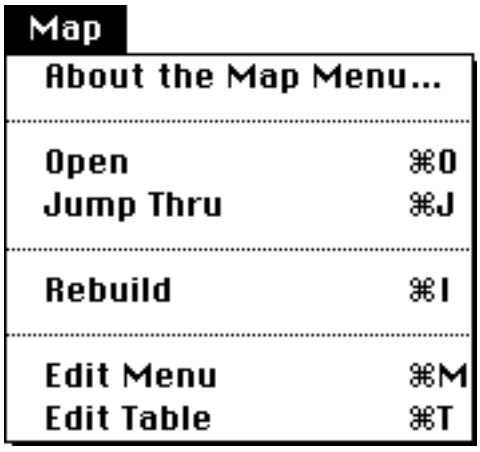


Figure 437. Map Menu

About the Map Menu... opens a word processing text document explaining this menu and its use.

Open, or Command-O, will open the outline window displaying the Object Database map (Figure 438) if it is not already open. If the window is already open, this item will bring the window to the front and select it.

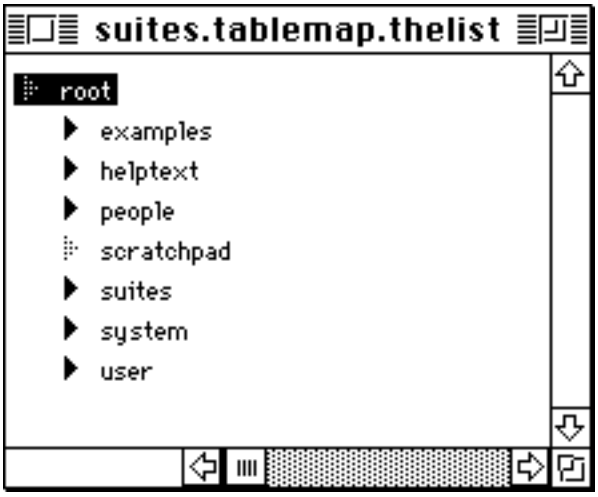


Figure 438. The Object Database Map Window

Jump Thru, or Command-J opens the editing window for the object you have selected in the Object Database map window and brings it to the front, enabling you to view and edit the object. If the map window is not frontmost, this option opens the window and brings it to the front but does not execute the jump.

As you edit the Object Database directly and through scripts, its contents change. You can rebuild the Object Database map so that it matches its contents by choosing Rebuild, or Command-I, from the Map menu. This process can take quite a while, so we don't recommend doing it routinely or when you are pressed for time.

The last two items on the Map menu Edit Menu and Edit Table are standard entries for a suite menu; see Chapter 6 for a discussion of this topic.

Glue Menu

The Glue menu appears on the Frontier menu bar in response to the user's selection of Commercial Developers from the Suites menu. This menu and its use are more closely related to the development and publication of Frontier-aware applications than to the use of the basic menu system. These subjects are therefore deferred to Chapter 10.

Navigating in and Using Frontier's Windows

As we have seen, Frontier includes several different types of windows. In this section, we will talk about how to navigate in and use each of these window types. We begin with table windows. Then we discuss word processing windows. Next, we describe the use of the three types of outline-based windows: outline windows, menubar editors, and script-editing

windows. Finally, we discuss the Quick Script window and the script-execution/debugging window.

We will discuss navigating within these windows and we will also cover how to use the windows in ways that are unique to specific types of windows.

In general, these windows are like other Macintosh windows with which you are undoubtedly familiar. You can scroll them, zoom them, close them, resize them, and so forth. You can also use the Find command from the UserLand menu (described above) to locate a particular piece of information in the Object Database.

Contents Page | Previous Section | Next Section -- **Table Windows**

HTML reformatting by Steven Noreyko January 1996

Chapter 4: continued

Table Windows

Table windows are in many ways at the heart of Frontier. The Object Database, which as you know is the repository of everything Frontier needs, is a collection of hierarchical table structures. Each table can contain any type of Frontier object, in any combination, including other tables.

Navigating in a table window is straightforward. You can type the first letter of the name of the object you are seeking and Frontier will take you to at least the vicinity of the object you seek. You can type more than one letter and Frontier looks for an object beginning with all of the characters you type.

To select an object in a table window, click on its item marker.

To delete an object in a table window, select it and then use an appropriate Edit menu command or keyboard equivalent to remove it.

To add an object to a table window, you can either use the Table menu option for the type of data you wish to add or you can follow this process:

1. Press Command-Return in the table window to which you wish to add the object. (It doesn't matter where in the table you do this.)

2. Type the name of the new object.

3. Click on the Kind: popup menu in the lower left corner of the window. This produces a popup menu like the one shown in Figure 439, which contains the same datatype options as the Table menu with the exception of the special scalars.

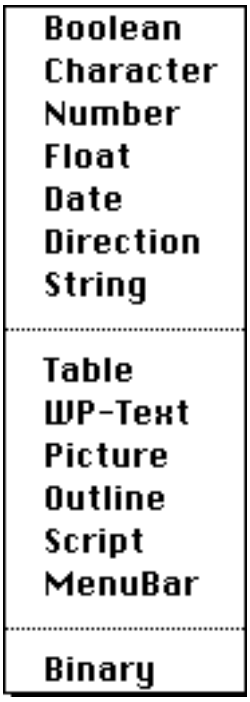


Figure 439. Popup Kind Menu in Table Window

4. Select the type of data you wish to add to the table.

5. If the object is a scalar and you wish to do so, you can press the Tab key to get to the Value column and give the object a value. Otherwise, you will probably want to double-click its item marker to open its window.

If you enter a new item into a table, you can select a non-scalar type for it without having to use the Kind popup. Just double-click on its item marker. A dialog box (see Figure 440) will appear. You can pick any of the available non-scalar datatypes from this dialog and Frontier will open an appropriate editing window. You can also name the object as part of this process.

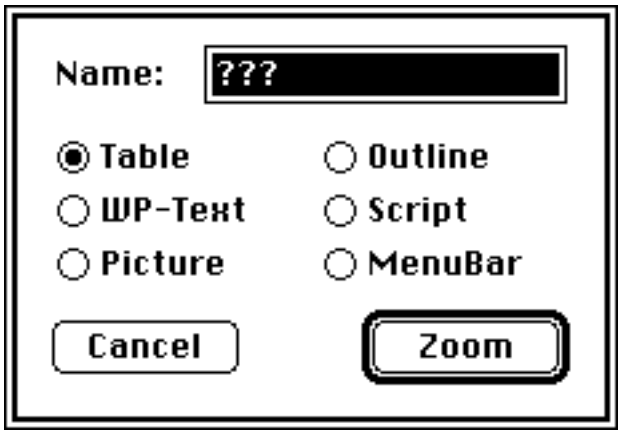


Figure 440. Dialog for Selecting Kind of Data for New Table Entry

WP Windows

Frontier's word processing text windows work very much like most word processors and

text editors you might have worked with on the Macintosh. They have a ruler you can hide and show with Command-R or with the appropriate menu item on the WP menu. You select text by double-clicking on a word or triple-clicking to select a paragraph. You can extend a selection by positioning the cursor where you want the selection to end and holding down the Shift key while you press the mouse button.

Styling and formatting of text are handled through the ruler line and through the last five options on the Frontier Edit menu.

To navigate in a word processing window, you can use the arrow keys to move up and down a line at a time or right and left a character at a time. Table 41 summarizes how the Command and option keys modify cursor movement.

Table 41. Cursor Movement Modifier Keys in Word Processing Windows

Modifier Key	Left Arrow	Right Arrow	Up Arrow	Down Arrow
Command Key	top of doc	bottom of doc	top of doc	bottom of doc
Option Key	left one word	right one word	left one word	right one word

Inserting new text into a word processing text document is a simple matter of positioning the cursor where you wish the new text to appear and typing the new text. Of course you can use all of the Macintosh's standard Cut/Copy/Paste operations as well.

Contents Page | Previous Section | Next Section -- **Outline-Based Windows**

HTML reformatting by Steven Noreyko January 1996

Chapter 4: continued

Outline-Based Windows

There are three outline-based windows in Frontier: outline windows, menubar editing windows and script-editing windows. We'll first discuss outline windows since everything that pertains to them is also relevant to the other two types. Then we'll talk about the individual distinctions of the other types of windows.

Frontier outlines differ from most computerized outlines in that they permit you to have as many summit-level headings as you like while many outliners limit you to just one summit heading.

A Frontier outline contains only text-based headings, not pictures or other types of data. Headings do not word-wrap, no matter how many characters they contain. Each heading in an outline is limited to 255 characters.

You add a heading to a Frontier outline by pressing Return. The new heading is inserted immediately below the one on which you are positioned. If you want to make it a child of the current heading, you can use the Tab key.

If the heading on which you are positioned has any expanded sub-headings, the new heading is inserted as the first child of the heading. If it doesn't have any sub-headings, or if its sub-headings are all collapsed, the new heading is inserted as a heading at the same level as the one on which you pressed Return.

To delete a heading, you select it by clicking on its item marker and then deleting it using the Clear command.

You collapse and expand a heading by double-clicking on its item marker. If the heading is already collapsed, it expands. If it is expanded, double-clicking its item marker collapses it. Collapsing a heading always hides all of the headings subordinate to it, including sub-headings of nested sub-headings, and so on. Expanding a heading, however, only expands the next level down. With the Outline menu (discussed above), you can fully expand and collapse an outline.

An outline can be in one of two modes for editing and navigation purposes. In text mode, the entire heading is not selected. The cursor is either positioned between characters or it selects multiple characters. In that mode, navigation follows the same rules as word processing documents described above. When the outline is in outline mode, however, the entire heading is selected by the bar cursor. You can switch into and out of text mode in an outline with the Enter key.

Table 4-2 describes how the Command and Option keys control navigation in an outline when it is in its default, structured mode.

Table 4-2. Cursor Movement Modifier Keys in Outline Windows

Mode	Key	No Modifier	Option	Command
Outline	up	structured up	flat up	up to first sibling
	down	structured down	flat down	down to last sibling
	left	flat up	structured left	up to first heading
	right	flat down	structured down	down to last heading

NOTE

You can switch an outline between structured mode (which is its default condition) and flat mode with the UserTalk verb op.flatCursorKeys. In that event, the arrow keys work differently from the way they work in structured mode. See the UserTalk Reference Guide for details.

In Table 42, the following terms have the indicated meanings:

- structured up up to previous sibling; beep if at first sibling
- structured down down to next sibling; beep if at last sibling
- structured left left to parent; beep if on a summit
- structured right right to first subhead; beep if no expanded sub-headings
- flat up up to previous heading; beep if on first summit
- flat down down to next heading; beep if on last expanded heading

In addition to the normal Cut/Copy/Paste operations, you can also reorganize a Frontier outline in one of these ways:

- dragging outline headings
- using the Tab and Shift-Tab key combinations
- using promote/demote operations
- using special Command-key combinations

The Frontier outliner also supports hoisting and de-hoisting operations which allow you to look at only part of an outline as if it were the entire outline.

If you click on a heading's item marker and hold the mouse button down, the cursor changes into a small hand that appears to be grasping the item marker. You can physically drag this item in the outline. Other item markers will turn into arrows pointing up, down, or on a 45-degree angle down and to the right indicating where you are about to place the outline segment. Figure 441 shows a selected heading being dragged so that it appears as a new summit. Notice that the item marker of Summit 2 is a down-arrow indicating that the user is presently moving the heading so that it is directly under and on the same level as Summit 2.

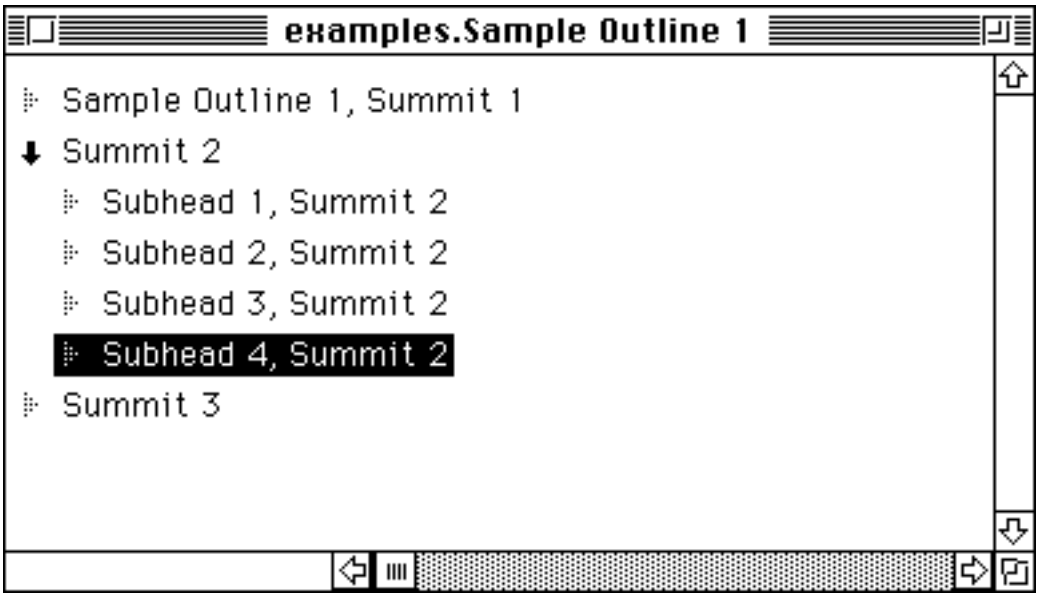


Figure 441. Moving a Heading by Direct Manipulation

If you press the Tab key while you're on a heading in outline mode, Frontier moves that heading to the right if possible. Shift-Tab moves the heading to the left if possible. If either movement is blocked because the heading on which you are positioned is already as far right or as far left as it will go, Frontier beeps.

NOTE

The ability to use the Tab key for reorganizing an outline can be turned on and off with the UserTalk verb op.tabKeyReorg. See the UserTalk Reference Guide for details.

You can choose the Promote and Demote options from the Outline menu to reorganize part of an outline as well. Each of these options works on the sub-headings of the current heading. If you promote sub-headings, you move them all to the left one level. If you demote them, you move the siblings below the selection to the right one level.

Frontier includes four fast reorganization key combinations. These are summarized in Table 43.

Table 43. Fast Reorganization Key Combinations

Command Key	Combination Effect
Command-U	Moves selected headline up, swapping with its previous sibling
Command-D	Moves the selected headline down, swapping with its next sibling
Command-L	Moves the selected headline to the left, making it the next sibling of its parent
Command-R	Moves the selected headline to the right, making it the last child of its previous sibling

HTML reformatting by Steven Noreyko January 1996

Chapter 4: continued

Outlines in the Menubar Editor

This section describes how working with the Frontier Menubar Editor differs from working with ordinary Frontier outlines.

The use of and navigation in a menubar editor is identical to that of any other outline. The differences you should note about the menubar editor are:

- A sub-heading is actually a nested sub-menu. We recommend you not nest menus more than two levels deep but we've seen a few three-level menus that work in particular settings. (The Macintosh system software will not permit you to nest menus deeper than five levels.)
- If you type a single hyphen as the entry in a heading, Frontier expands that to a full menu separator.
- You can examine the script connected to a menu item by selecting it or positioning your cursor in it and then clicking on the Script button.
- You can see which Command-key equivalents have been set for this menu and set up your own from the Cmd: popup menu in the lower left of the window.

Script-Editing Windows

As with menubar-editing windows, script editors operate nearly identically to other outline windows in Frontier. Here are the other considerations you must remember when dealing with a script editor:

- Toggling comments with the menu or keyboard equivalent can change the nature of headings embedded under the heading whose type you change.
- To insert a comment line as you type in a UserTalk script, press Return while you hold down the Shift key.
- Command-clicking on an item marker toggles a breakpoint on that line.

Script-Related Windows

Aside from the script editing window discussed in the previous section, there are two other Frontier windows involved directly with scripting. The Quick Script window is where you can type and immediately execute short scripts. The script execution window provides a place for you to work interactively with a running script to understand and debug it. In this section, we'll look at each of these windows.

Quick Script Window

You can type scripts into the Quick Script window and execute them immediately by

pressing the Enter key or by clicking on the Run button. One-line scripts can be entered exactly as they are in a script-editing window. Multi-line scripts must, however, follow two additional conventions:

- Wherever you would normally place a carriage return in a script (i.e., when starting a new line or heading), you must use a semicolon in the Quick Script window.
- Wherever you would use a Tab key to indent a line of code in a script, you must use curly braces in the Quick Script window around the lines that would be indented in a script outline. In this case, you must not add a semicolon to the preceding line.

Figure 442 shows a small sample script as entered in a script editing window. Figure 443 shows the same script as it would be formatted in the Quick Script window.

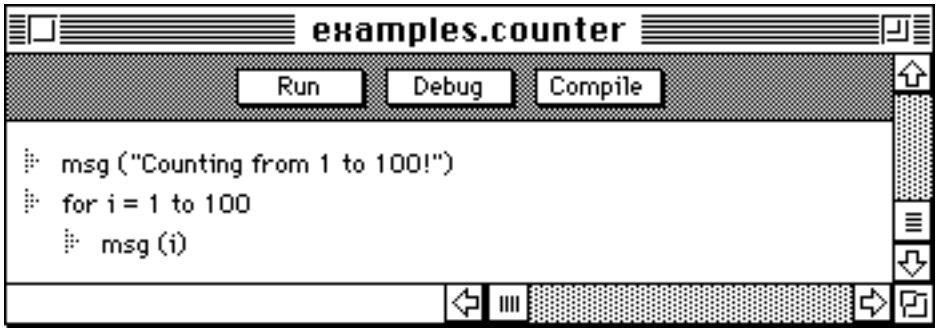


Figure 442. Sample Script in Script-Editing Window

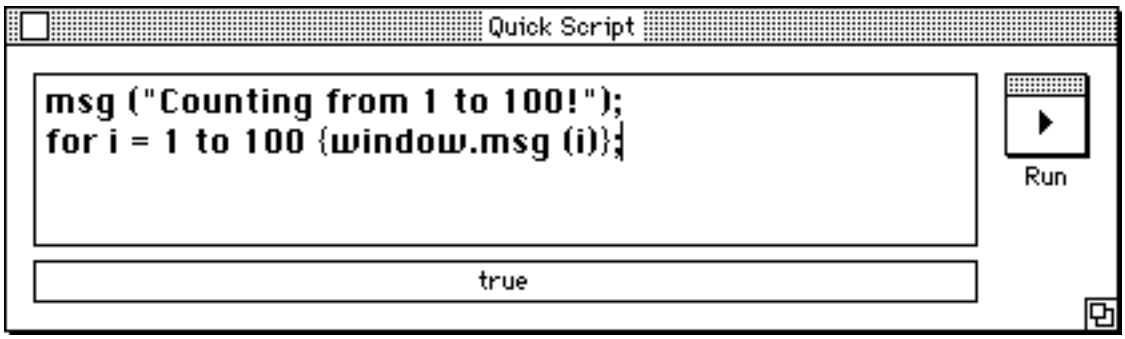


Figure 443. Sample Script in Quick Script Window

Notice that there is no semicolon after the line that starts the for loop. That's because the loop is treated as a single construct in the interpreter.

Script-Execution Window

When you have entered a script into a script-editing window and you wish to run it from there, you can do so just by clicking on the Run button. This approach will only work with scripts that don't require parameters, of course. Even with such scripts, if they begin with the keyword on (see Chapter 6), you'll have to include a command at the end of the script to run it. For example, if a script is named countTables and you start it out like this:

```
on countTables( )
```

then you'll need a line like this:

```
countTables( )
```

at the end of the script and outside of its indentation scope or you won't be able to use this Run approach to script execution.

If you want to debug your program whether that means simply tracing through its execution to understand it better or looking for an error of some sort you can click on the Debug button in the script execution window. That will display debugging options (see Figure 444).

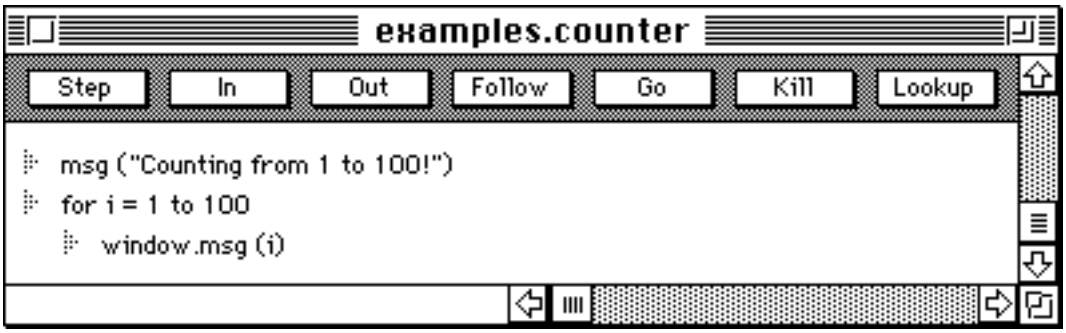


Figure 444. Debugging Options in Script Window

This window's buttons and their use in debugging are described in detail in Chapter 6, so we won't go into it now. If you click on Debug when you meant to click on Run, just click on the Go button in this window to start execution or on Kill to go back to the script editor.

HTML reformatting by Steven Noreyko January 1996

Chapter 5

The Object Database

This chapter describes the Frontier Object Database in detail. We have encountered the database in earlier discussions, particularly in Chapters 2 and 3. In this chapter, we'll examine it closely to see:

- how it's organized
- how to address objects stored in it
- how to find objects stored in it
- what it means to create a new variable with respect to the Object Database
- how to export information from it and import information into it
- how to use it for personal information and make it your own

Organization of the Object Database

The Object Database is a table that can hold other tables or pieces of other types of information. Frontier understands how to deal with a wide range of datatypes. These are detailed in the UserTalk Reference Guide. The most common and important ones are defined on the "Kind" popup menu in the Object Database editing table. They are:

boolean
character
number
floating-point number
date
direction
string
word processing text
picture
outline
script
menubar
binary

All of these datatypes have specific characteristics such as size, range of values they can represent, and so forth. These are described in detail in the UserTalk Reference Guide.

The first seven of these datatypes are scalar values. You can edit their contents directly in the table window in which they are located. Word processing text, picture, outline, script, table, and menubar values, on the other hand, are complex values that require their own special editing windows. You can tell a scalar from a complex Frontier datatype by examining their entries in the Object Database. A scalar object's item marker is always gray.

A complex object's item marker is always a full triangle which is solid black if it is the currently selected object in the table, outlined if it is not.

Double-clicking on the item marker of a complex Frontier object opens the editing window for that object and makes that object the current selection in its table.

Navigating in Object Database windows was discussed in Chapter 4.

Purposes of the Database

The Object Database in Frontier serves a number of purposes. Viewed from a programming perspective, you could think of this database as the symbol table for your Frontier scripts - an extensible hierarchy of global variables. Like any programming language, UserTalk needs a symbol table. Frontier's Object Database makes an excellent place to store this symbol table.

Of course, it is much more powerful than the symbol tables associated with other programming languages with which you may be familiar. There are at least three important differences between the Object Database and a traditional symbol table.

The Frontier Object Database:

- stores contents permanently
- is hierarchic
- can be accessed interactively, from a script or from an external program written in a conventional programming language like C or Pascal

Organization of the Database

Figure 5-1 shows you the top part of the hierarchy that makes up the Frontier Object Database as the database is delivered by UserLand. As you use this database, import scripts and other objects into it, and otherwise customize it, this organization changes, of course.

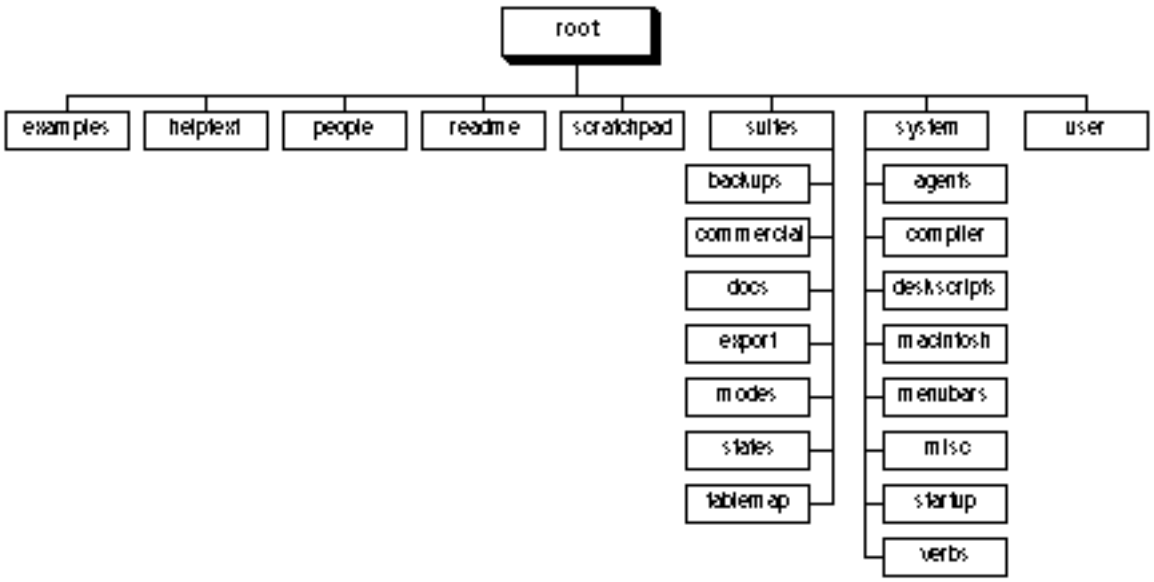


Figure 5-1. Partial Object Database Hierarchy

As you can see, the base table in the Object Database is called root. It has eight items in it: seven tables and one word processing text entry (the "readme" object). Each of the seven tables, in turn, has one or more elements, some of which may be further tables. This nesting of sub-tables can continue up to 25 levels deep in the database, though as a practical matter you'll probably find that objects nested deeper than 7-10 layers may become cumbersome to use.

Contents Page | Next Section -- **Sub-Tables in root**

HTML reformatting by Steven Noreyko January 1996

Chapter 5: continued

Sub-Tables in root

You probably won't have much occasion to make changes to most of the tables that make up the Object Database. Later in this chapter, we'll take a look at some places where you will wish to make such changes. For now, let's look at the tables stored in the root table of Frontier.

Examples

The examples table holds several dozen examples of objects and scripts that are used extensively in this manual and in the UserTalk Reference Guide. You should feel free to add your own examples to this table as you gain experience with Frontier.

Helptext

Contained in the helptext table are a number of outlines and word processing text objects whose purpose is to provide context-sensitive help with various operations in Frontier. Many of these objects are opened in response to your actions at various points in Frontier. But you may find them useful to open directly as well. For example, double-click on the item marker next to datatypes and you'll see a window something like Figure5-2. This is a list of all the datatypes in UserTalk.

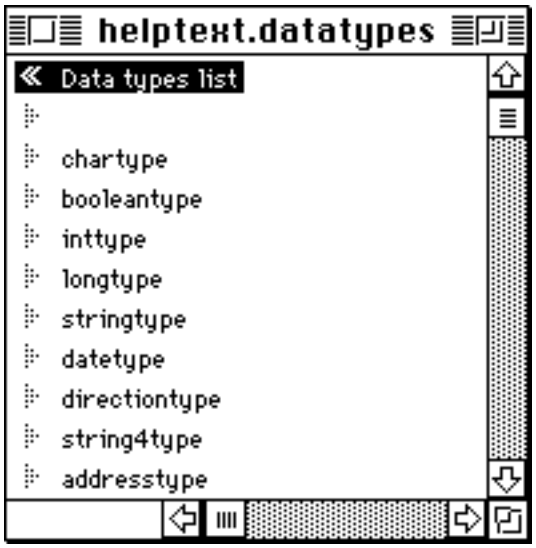


Figure 5-2. Partial Listing of root.helptext.datatypes Outline

If you ever wonder about a specific datatype, you can always jump to this table (helptext.datatypes) by pressing Command-J and then entering its name into the dialog. Frontier will open the outline as shown above (or however you left the window when you

last closed it).

People

The people table is one you'll see frequently. When you run Frontier for the first time, it asks you for your name and initials. Among other initialization it performs at that point, Frontier creates a new entry in the people table with your initials. For example, if your name is Xenophanes Xerxes Xenophon, and if you enter your initials as "XXX," Frontier creates a table entry called people.XXX. As a script writer, you might find it convenient to use this known location for storing certain kinds of information your scripts need when they are running on a target user's system.

Be careful about writing scripts that make explicit reference to your people table. Such scripts will not be usable by others with whom you might wish to share them. Because your people table is in a path that Frontier has registered in its paths table, you can always omit the people.XXX portion of the address and use just the cell name. This approach enables everyone with Frontier to use your scripts.

Readme

As you can undoubtedly tell from its name, the readme entry in the root table is a place where last-minute information about Frontier - data that was too late to make it into this manual or in the release notes that may accompany Frontier - is placed. We encourage you to open that table entry and peruse it.

Scratchpad

You'll probably find frequent use for the scratchpad table, particularly during development and testing of reasonably complex Frontier scripts and suites. We use this location in some of the example scripts in the documentation as well. As you can tell from its name, it is intended as a somewhat temporary storage area, although like all Frontier table entries, it is saved to disk and remembered from session to session.

Suites

One of the most interesting and powerful ideas in Frontier is that of the suite. We cover this subject in some depth in Chapter 6, but you should know that suites are collections of related scripts that share a special menu. Figure 5-1 (see page 79) shows the suites that come with Frontier; you'll undoubtedly find yourself making use of this table, adding suites to it over time.

System

The system table is one of the most important sub-tables in Frontier. As you can see from Figure 5-1, it includes a number of useful-sounding entries. For example, Frontier includes the ability to create scripts that run as background tasks on a scheduled basis. These scripts are called agents and are stored in system.agents. Desktop scripts often begin their lives in system.deskscripts, though that is purely an optional design. All of the verbs in UserTalk are defined in system.verbs and its many sub-tables. If you peruse this sub-table, particularly the sub-table called builtins, you may be surprised to find that a great many UserTalk verbs are defined in scripts that use other verbs in the language. Whenever you see

a reference to the "kernel," you are looking at a direct call to the Frontier program.

Information On Each Object

Every object in the Frontier Object Database, regardless of type, has three pieces of information stored, corresponding to the three columns in the table:

- name
- value
- kind

Every object must have a name. You normally use the name of an object to access its contents, although as we will see in the next section you can use its relative position in the table when that information is known and useful. A Frontier object's name must not be longer than 32 characters. While most of the names you'll see in the Object Database are one word, this is not mandatory. Multi-word names are perfectly legal. As we'll see in the next section, they present special addressing problems for scripts but nothing insurmountable.

What is displayed under an object's value depends on two factors: the kind of data involved and, if it is non-scalar, whether it has been loaded from the disk during the present session. For example, take a look at Figure 5-3, which is part of the root.examples table as it is shipped by UserLand.

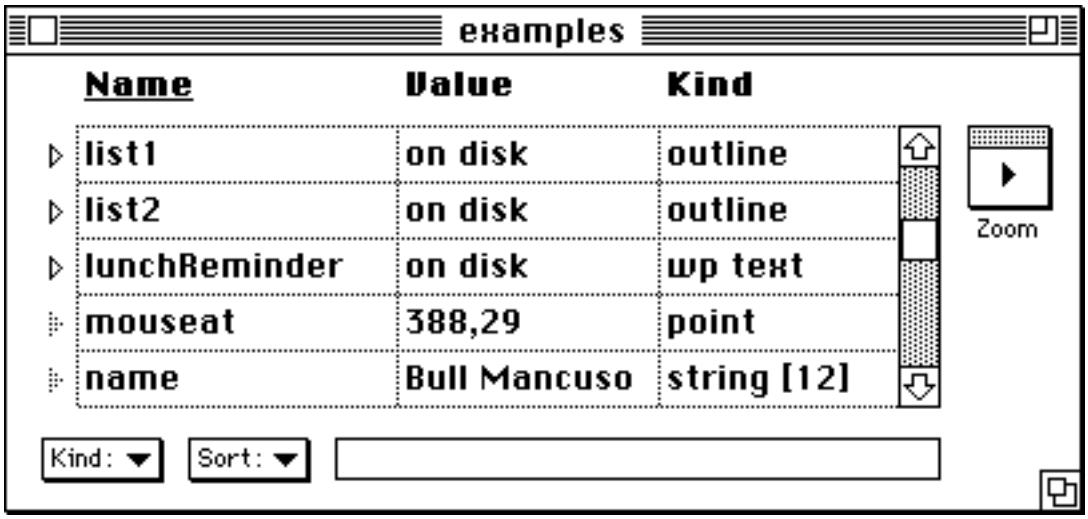


Figure 5-3. Partial Listing of examples Table

Notice that the objects called list1 and list2 are outlines and are stored on disk. The object lunchReminder is a word processing text object that is also on disk. But mouseat is a scalar object, a point, and its value is reflected directly in its entry in the table. Similarly, name is a scalar object, a string, so its value is shown in the table.

The outline object called list1 is, as we have seen, stored on disk. Double-click its item marker to open it. Now look at its entry in the examples table. It indicates the contents of the outline as far as Frontier knows them at this point, namely that the outline has five lines.

HTML reformatting by Steven Noreyko January 1996

Chapter 5: continued

Addressing Objects

As with any database, the objects in the Frontier Object Database are useful only to the extent that you can access and, as appropriate, modify their contents. In other words, you need a means of addressing objects in the database, both to retrieve their contents (possibly for use in a UserTalk script or for display) and to change their contents.

The basic method of addressing an object in Frontier is by supplying its full name, with each sub-table of which it is a member appearing in order from the root level down to the object, and each sub-table's name separated from the others by a period. Thus the sub-table called examples can be addressed as root.examples. An entry in that table called name would be addressed as examples.name. (Notice that we omit the "root" from the address. You never need to type "root" explicitly because Frontier always understands you to be addressing objects that have their ultimate home in the root table.)

If you type the address of a database object into the Quick Script window, Frontier will tell you that object's value. (Remember from above that the value of an object depends on the type of object and whether it is stored on disk or not.) For example, Figure 5-4 shows the results of typing root.examples into the Quick Script window. Since examples is a table, its value is the number of items it contains.

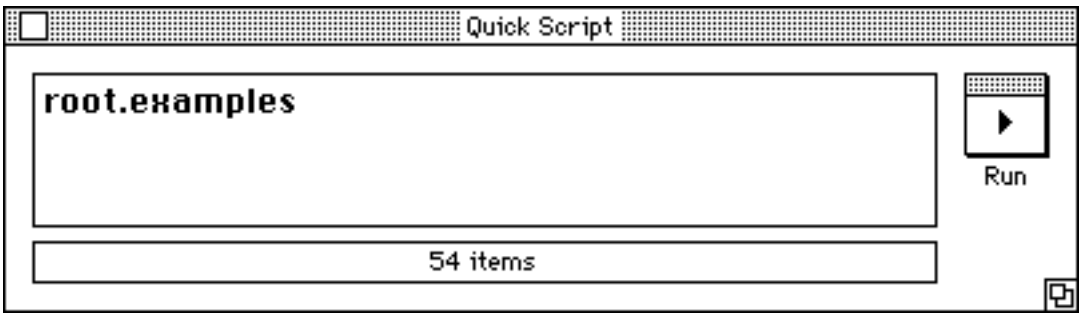


Figure 5-4. Typing an Object's Address Produces its Value

A complete understanding of addressing objects, however, requires that we discuss several other subjects:

- names containing special characters, such as spaces
- omission of paths that are "known" to Frontier
- using an indexing method of addressing

Names Containing Special Characters

In the table root.examples, there are three sample outlines that are used extensively in the

UserTalk Reference Guide. They are called "Sample Outline 1," "Sample Outline 2," and "Sample Outline 3." If you try to type the name of one of these objects into the Quick Script window, an error condition (see Figure 5-5) will result.

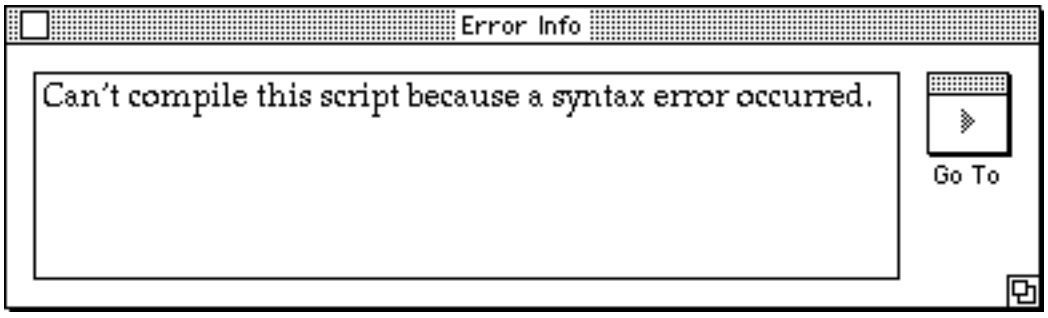


Figure 5-5. Syntax Error from Using Multi-Word Object Name

If you click on the "Go To" button in this window to ask Frontier to take you to the place where the error occurs, you'll find the cursor flashing after the word "Outline." UserTalk sees this text as a script containing three consecutive table identifiers - "Sample", "Outline", and "1" - with no operators between them. This is an illegal script.

To access such objects in UserTalk, you must enclose the multi-word name in quotation marks and then you must enclose the entire quoted string in square brackets (see Figure 5-6).

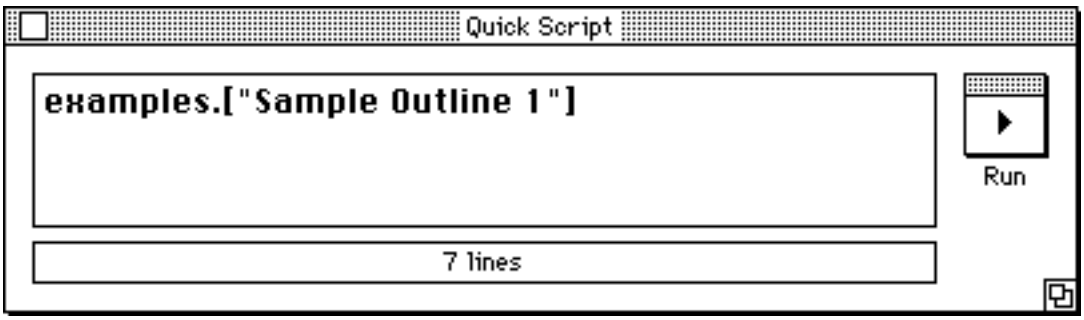


Figure 5-6. Proper Syntax for Multi-Word Object Name

This syntax is actually a special case of the use of square brackets to force UserTalk to evaluate the expression contained between them and to use the result of that evaluation as an identifier. This somewhat advanced technique can be used to create generic scripts that would otherwise have to be customized for a particular user or configuration. For example, if you want to address the table pointed to by the user's entry in the people table but you don't know the user's initials, you can use a line like this:

```
@people.[user.initials]
```

This line instructs UserTalk to evaluate the expression user.initials (the Object Database location where the user's initials are stored) and to use the result as the name of the table item. (The commercial "at" sign (@) is the address-of operator, which is discussed in Chapter 6.)

When You Can Use Partial Object Names

As we saw earlier, you can type an object's name into the Quick Script window and retrieve its value. Let's look at an example. Type the following line into the Quick Script window and execute it:

system.misc

You'll probably get an answer like "6 items" in the message area of the Quick Script window. This is hardly surprising. But now remove the first word of the address and type just "misc" into the Quick Script window. You'll get the same answer. (You might even want to be a real skeptic and change the previous contents of the message area by entering something like "2+2" into the Quick Script window between these two tests.) How does Frontier know where to find the object called misc if you don't tell it?

For the answer, type the following line into the Quick Script window and execute it:

```
edit (@misc.paths)
```

The result should look something like Figure 5-7. Examine this table and you'll quickly discover that it is a series of pre-defined partial paths to objects in the Frontier Object Database. Any object whose full name begins with any of these partial paths will be found if you supply only the remainder of its name (i.e., the part that would be appended to the end of one of these known paths).

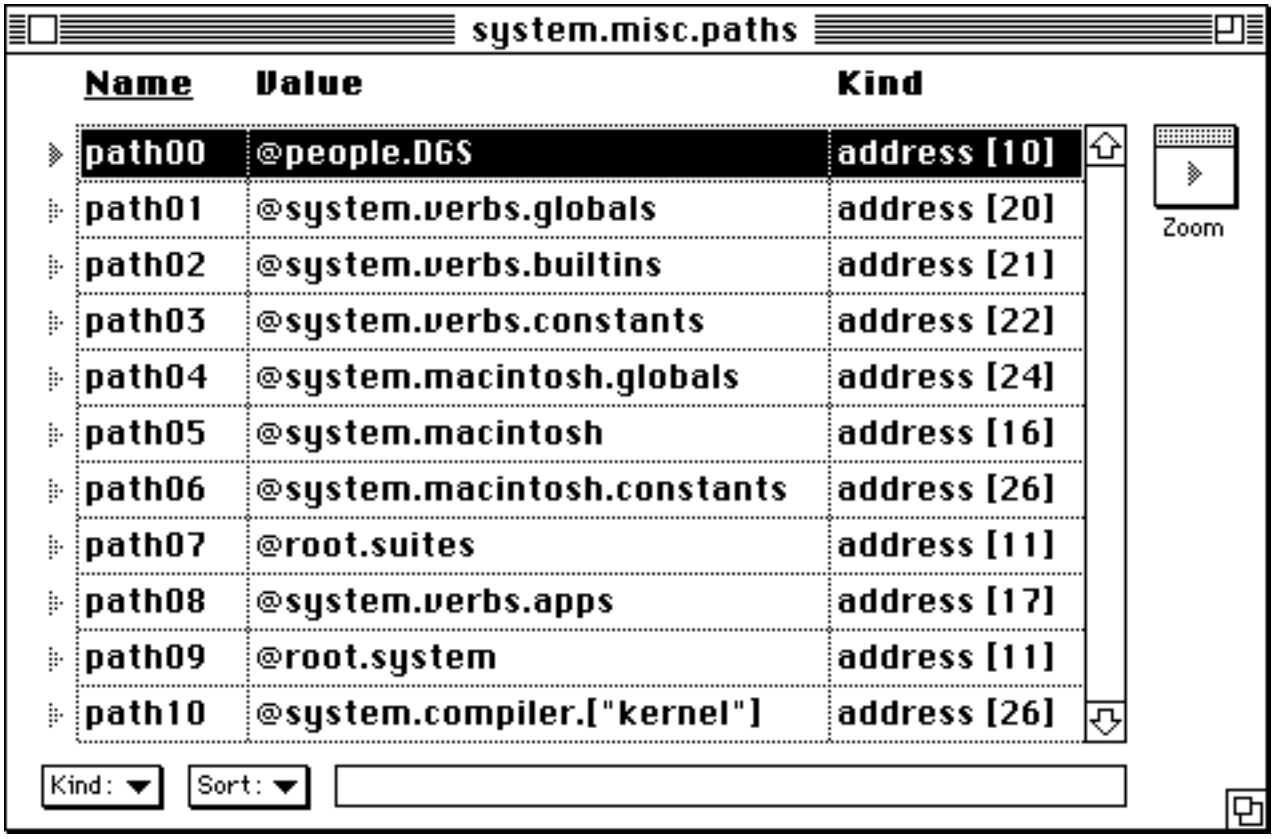


Figure 5-7. Table of Pre-Defined Object Database Partial Paths

As you have probably guessed, you are free to add to this table as you need to do so for programming convenience or for ease of use in delivering your Frontier applications. Frontier searches this table from top to bottom, so duplicate names are resolved quite easily.

If there were two objects called misc, one in people.DGS and one in root.system, the one in the people table would essentially override the other unless a complete path to the latter was provided.

Contents Page | Previous Section | Next Section -- **Indexed Addressing Alternative**

HTML reformatting by Steven Noreyko January 1996

Chapter 5: continued

Indexed Addressing Alternative

The objects in a Frontier Object Database table can be referenced by their relative positions in the table. The first entry in the table can be addressed as:

```
table.name[1]
```

All other entries are similarly addressed. The 12th item is addressable as:

```
table.name[12]
```

This approach is most often used when scripting the Frontier environment. It permits you to deal with a Frontier table exactly like an array in other programming languages. A practical example of its potential use can be seen by examining the table system.verbs.constants. This table has 12 entries, one for each month of the year. If you sort this table by name (which is its default condition as it is shipped by UserLand Software), you can access the month by its index number, as in the example shown in Figure 5-8.

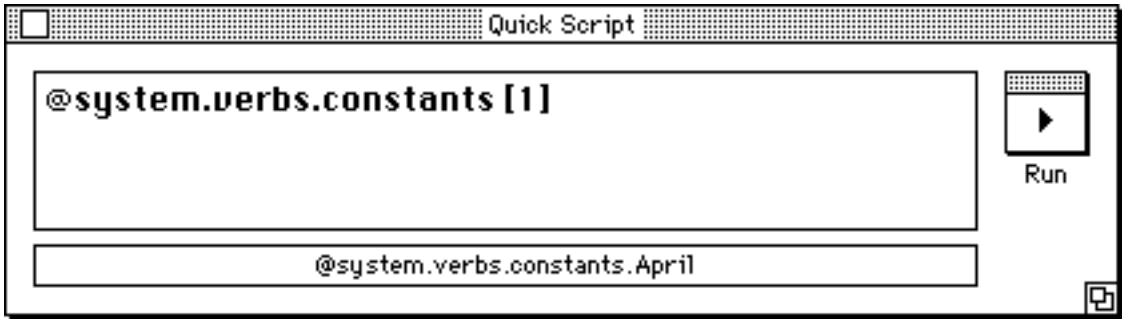


Figure 5-8. Accessing a Month by Index Number

Finding Objects in the Database

There are two levels at which you might want to look for information stored in the Object Database. You might wish to locate a specific object - table, script, scalar, or other Frontier object - in its table. Or you might want to find the contents of an outline or word processing text entry. Frontier supports both types of data-locating capabiliites.

Locating an Object

Other than manually opening tables and sub-tables until you find an object, you can locate a Frontier object in two ways. We've seen these before, but it won't hurt to point them out again since finding things in Frontier is something all UserTalk script writers find themselves

doing a lot.

The fastest way to locate an object in Frontier is with the Jump command from the UserLand menu, also accessible by Command-J from the keyboard. This command brings up a dialog into which you type the name of the object you are looking for. If the object exists in Frontier and you've supplied its correct name, you'll be taken directly to it. If the object is a scalar, its entry will be selected. If it's a complex object with its own editing window, that window will be opened. If Frontier can't find the object, it beeps.

If you are looking at some information in a script (or any other object, although it is most often useful with scripts), you can hold down the Command key and double-click on any Frontier term and be taken directly to its definition if it can be found. When scripts call each other, this is often the quickest way to navigate from a script to another script it calls.

Finding Text Information

As we saw in Chapter 4, Frontier includes built-in find-and-replace capability. You can use this power to locate any piece of information you want to find in the Object Database. Recall that the Find & Replace dialog lets you determine whether your search is case-sensitive, whether to confine the search to whole words or not, and whether to wrap around to the beginning of the portion of the database you are searching if the information you want isn't found before the end of the table or scope is reached.

How much of the Object Database is searched for your term? That depends on what window is frontmost when you invoke the operation. Table 5-1 summarizes the scope of the Find & Replace operation.

Table 5-1. Find and Replace

Find & Replace Invoked In...	Has Scope Of...
Main Window	Entire Object Database
Outline, Script, Menubar,	That window, except in a Menubar or Word Proce window where it also includes all of its scripts
Table Window	Table and all of its sub-tables, and so on until l sub-table is searched

Local Variables and the Database

You can create new entries in the Object Database in several different ways. The manual method for doing so was described in Chapter 2.

From the Quick Script window, you create a new variable in the Object Database simply by assigning it a value. For example, scratchpad.y does not exist when you receive Frontier from UserLand Software. But if you type this line into the Quick Script window, you can then examine the scratchpad table and confirm that it has an entry called y with a value of 13.04 and a type of double:

```
scratchpad.y = 13.04
```

In UserTalk scripts, you create a new entry in the Object Database every time you declare

or use a local variable. This entry is not in a part of the Object Database that is permanently stored. It is stored in the dynamic stack maintained by Frontier during script execution. When you are debugging a script, this location is accessible through the Object Database structure at location system.compiler.stack.

In your UserTalk scripts, you should get into the habit of defining variables by using the local keyword, which is discussed in detail in Chapter 6.

Importing to and Exporting from the Database

Information that exists in the Frontier Object Database, including UserTalk scripts, can be exported. Conversely, information exported from one copy of Frontier can be imported into another. In this section, we'll discuss exporting items as objects and as desktop scripts (which are discussed in detail in Chapter 3). Then we'll talk about importing objects from these exported files. Finally, we'll talk about moving information between Frontier and other applications which lack IAC support or whose IAC wires do not support information transfer operations.

Exporting Objects

You can export any Frontier object to the desktop for import to another copy of Frontier or back into your original Frontier root file. We discussed the UserLand menu options associated with this process in Chapter 4.

When you export an object using one of the options from the "Export an Object..." item on the UserLand menu, Frontier saves it in the folder you indicate, with the name you provide for it. In the process, Frontier assigns it a creator and file type that permit it to be recognized by any copy of Frontier.

Desktop scripts, which we examined in Chapter 3, are exported in a similar way to other objects, but are stored in a special format on the disk. When they are double-clicked by the user, they are simply executed rather than being imported into Frontier. (Frontier must be available before desktop scripts can execute, however.)

You can also copy Frontier objects and paste them into other applications via the Clipboard, just as you are used to doing with other Macintosh applications. You can select text from a word processing text object, for example, copy it to the Clipboard with the Copy command from the Edit menu and then paste it into a document in another application. You can do the same with individual headings of Frontier outline objects, with scalars, and with individual names of table entries.

There is also a more powerful way to copy Frontier objects. In a table window, you can select an outline or word processing text object's entry and, without opening, it, choose the "Copy" or "Cut" option from Frontier's Edit menu. Now when you move to another application, you can paste the entire document or outline - including outline indentation but not the item markers in an outline. In the process of such data transfer, you lose formatting such as bold text, fonts, and margins, just as when you copy text from any word processor into another application.

This Clipboard approach to moving information from Frontier to other applications extends to tables. You can select a sub-table's entry in a table window, copy it, and paste it into

another application document. If you do so, you'll find that the table objects are copied over as a single block of text, but each table entry is labeled with its Frontier name and datatype.

Importing Objects

You can import Frontier objects into any Frontier.root file. If the object was exported from Frontier to a file using the "Export an Object..." option from the UserLand menu, you can import it into a Frontier.root file one of two ways:

double-clicking on it in the Finder, with or without Frontier running (Frontier will be launched if it is not running)

from within Frontier, choosing "Open..." from the File menu and then choosing the file

If the object you wish to import into Frontier is a desktop script, you can hold down the Command key when you double-click its icon in the Finder and continue to hold the Command key down until the script's editing window opens in Frontier. You can then save the root file and the desktop script will be saved in the table system.deskscripts with the name of the script as it appears in the script's header.

When you open a desktop script while holding down the Command key, the script is loaded into your root file. If you don't save the root next time you exit Frontier, the script will not be a permanent part of the root file. In other words, you have to make a conscious decision not to save a desktop script if you open its script from the Finder as described here.

Using the Clipboard, you can bring text information from other applications into Frontier. Select some text in an application window, copy it, then switch to Frontier. Paste the text. The outcome of this process depends on what kind of window is frontmost, as shown in Table 5-2.

Table 5-2. Pasting Text into Frontier from Clipboard

Frontmost Window Type	Result of Paste Operation
Table	Text is pasted into new string object named "pasted text"
Word Processing Text	Replaces selection, if any, or is inserted at insertion point if no selection
Outline	In text mode (no heading selected), behavior is same as for Word Processing Text objects. Otherwise, pasted text is inserted into the outline with carriage returns and tab working as you would expect. The only exception is that if any single line of text exceeds 255 characters in length, it is broken into multiple segments of 255 characters each and segments become headings at the same level as the selected heading and immediately below one another.

Personal Use of the Database

Aside from its value and importance in scripting, the Frontier Object Database is a powerful construct in its own right. You might find the fact that it is a full-text retrieval database quite useful, for example, in storing such things as notes to yourself, electronic mail messages,

lists of electronic mail accounts, records of users of a local area network, outlines of projects, reminders about appointments, and a host of other items that may or may not be related to a UserTalk script.

You should feel free to make extensive use of the database for such things. We recommend that you use your people.XXX table (where "XXX," of course, stands for your initials) as the storage starting point. But you can build sub-tables, word processing text entries, and a host of other database entities in this table.

The only limit on what you can use this database for is your imagination.

We saw in Chapter 2 how to create new entries in an Object Database table. Using the Macintosh's Clipboard or UserTalk scripts, you can grab information from other applications and store it permanently in the Frontier Object Database where it can be easily retrieved and used later. You can timestamp entries and choose from outlines, word processing text documents, or scalar values for each piece of information while keeping all related data together in the table.

Experiment with it. Many of Frontier's early users became addicted to the database and came to depend on it.

HTML reformatting by Steven Noreyko January 1996

Chapter 6

Writing UserTalk Scripts

This chapter discusses the process of creating, storing, testing, debugging, and installing UserTalk scripts. It is divided into two major sections. The first covers the mechanics of scripting. The second discusses some key concepts involved in UserTalk scripting that may differ from your previous programming or scripting experience.

Scripting Mechanics

In this section, we will discuss how to create, edit, comment, and compile a script. We will also describe several ways of running scripts, including how to attach a script to a menu item. The section concludes with a discussion of debugging strategies for UserTalk scripts.

Creating a UserTalk Script

Like everything else in the Frontier environment, UserTalk scripts reside in the Object Database. The first step, then, in creating a new UserTalk script is to create an entry for it in the appropriate table in the Object Database.

Creating the entry is quite simple and straightforward. We've seen (Chapter 1) how to create entries in the Object Database manually.

You can also use a UserTalk verb to do so, as in this example:

```
new (scriptType, @people.DGS.newScript)
```

This would create a new script called newScript in the people.DGS sub-table.

Once you've created a new script entry in a table, you simply double-click its leader character and it opens. You can then type your script, resize, move, and close the window, and so forth.

Creating a new script entry is easy. Deciding where to put the script may require a bit more thought. Unlike your terribly neat desk, it is not necessarily true that there is a place for everything. You can often choose from among several good alternatives for places to store your UserTalk scripts. Here are some issues you should consider in choosing the right home for a particular script or suite of scripts.

If you're writing a collection of scripts that will share a single menu - in UserTalk parlance, a suite - you should store them in the suites table. You may also want to modify the Suites menu in system.misc.menubar to give the user a simple way to open your suite of scripts. It is not essential that your suites be stored in the suites table, but it will be easier for users to

deal with their Frontier environments if you make such decisions according to the model they've come to expect. UserLand suites are shipped in a special folder of the same name and are loaded into the suites table when the user indicates a desire to load them into Frontier.

If you are the only person who will use your scripts or suites, then, of course, you should feel free to store them wherever you like. It is, after all, your system. Experience has shown that storing such personal-use information in your people table has two distinct advantages. First, it makes it easier to remember where you put things when you try to find them six months later. Second, if a new version of Frontier is shipped and it has a new version of the root file, you won't have to rummage around your old root looking for things you don't want to lose when you upgrade.

Scripts that are designed to facilitate the use of an external application from Frontier are referred to as "glue" scripts. They should be stored in system.verbs.apps. (See Chapter 10 for a detailed discussion of such scripts.)

Editing a UserTalk Script

There is very little difference between editing a UserTalk script and editing any other Frontier outline. All of the normal Macintosh editing commands are available to you while you edit a script. This includes all of the options on the Edit menu. (Copying requires additional work in some situations; this is discussed below.)

Although UserTalk does not require it, we strongly recommend that you design your scripts so that they have only one summit and that this summit contain the name of the script preceded by the keyword "on." Here is how the header would look for a script called demoFormat:

This form of the script header is optional on scripts that will not be called from other scripts (including from scripts typed into the Quick Script window), but we encourage you to use it wherever there is not a good reason not to do so. A script like a menubar script, whose role is to drive other scripts, should not be formatted this way. Desktop scripts, on the other hand, must be formatted using this approach.

NOTE

If you format a script using this approach, and then you wish to debug it, you will generally need to add a line to the script that calls the local script that starts with the keyword "on." In the above example, you would add a line like this:

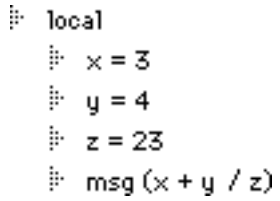
```
demoFormat ( )
```

Once debugging is complete, you should remove this line. Generally, you'll make this the last line in your script.

The structure of a script as reflected in its levels of indentation is significant. Your script may behave unexpectedly or fail to execute at all if you indent lines incorrectly. Levels of indentation in a Frontier script-editing window are used primarily to indicate flow of

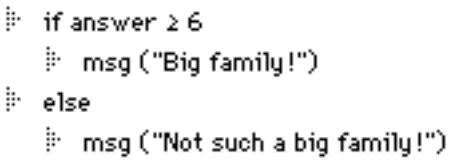
execution. For the most part, the UserTalk compiler will help you avoid obvious problems with indentation.

For example, if you try to compile the following script, UserTalk will indicate that it cannot compile it because of a syntax error:



Obviously, you didn't intend the last line of this script - which uses a UserTalk verb to place a message in the Main Window - to be indented as a local variable declaration.

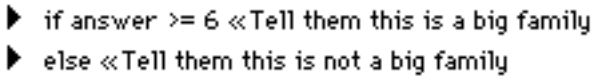
Normally, you use indentation with related groups of UserTalk statements such as if-then-else constructs, for and while loops, and the like. You can think of the indentation levels as serving the same purpose as the "end" statement in many other programming languages. Because indentation reveals structure, UserTalk does not need an end statement to show where a particular group of related lines ends. The syntax of these constructs is discussed in detail in Chapter 1 of the UserTalk Reference Guide. Let's look at some examples of how such script components are structured in UserTalk. Here's a simple "if" statement, properly formatted:



You can probably figure out that this script fragment looks at a variable called answer. If it holds a value of 6 or larger, the script displays one message and if the number is less than 6, it displays a different message. One of the real advantages of having an outline approach to scripting can be seen from the following fragment, which is actually identical to the one above except that the outline has been collapsed and comments have been added:

Users who don't want or need to understand detailed levels of processing in a script can look only at the level in which they're interested. The judicious use of comments (a subject we'll discuss later in this chapter) makes this easier.

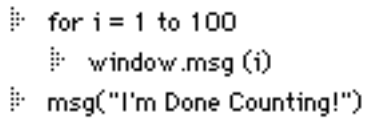
Here's a simple for loop, properly formatted:



Only the second line - the one that puts the numbers into the message area of the window in which the script is run - is executed 100 times because it is the only indented line under the "for" loop header.

If you wish to copy a UserTalk script, you should select the script's entry in the Frontier table in which it is stored by clicking on its item marker. Then use the Copy command to

copy its contents. You can then use the Paste command to place the script into another table or into a script-editing window.



Contents Page | Next Section -- **Comments in UserTalk Scripts**

HTML reformatting by Steven Noreyko January 1996

Chapter 6: continued

Comments in UserTalk Scripts

The UserTalk symbol for a comment is the chevron character (), made by holding down the Option key while you press the backslash key. All comments begin with this symbol.

Comments are easy to include in your UserTalk scripts and we encourage you to use them extensively. As with most modern programming languages, there are two kinds of comments: full-line comments and line-ending comments. Both begin with a comment character. In a full-line comment, the comment marker actually replaces the item marker in the outline where you are writing your script. A line-ending comment is one that follows some executable code in the same heading. It, too, begins with the comment character. In both cases, the UserTalk compiler stops trying to interpret programming information as soon as it encounters a comment character.

Because full-line comments are also part of the outline structure of a UserTalk script, all lines indented under a full-line comment are automatically comments and cannot be treated as executable code. If there are collapsed headings below a comment heading, the comment character is emboldened. Figure 61 shows a script fragment that demonstrates all of these facts about comments in UserTalk scripts.

Figure 61. Script Fragment Showing Comment Usage

There are several ways you can enter a full-line comment into a script or convert an executable line to a comment.

You can use the Script menu's "Toggle Comment" option to switch back and forth between a comment and an executable line of UserTalk code. Another way to enter a full-line comment is by pressing Shift-Return.

Frontier is quite intelligent about how it deals with comments and executable lines of code. For example, if you enter a line of executable code, and then indent it under a comment line

so that it becomes commented, then return that line to its previous level, it "remembers" that it is executable and toggles itself out of comment mode.

Compiling a UserTalk Script

All UserTalk scripts exist in two forms: source code and compiled code. The source code is stored in a script object in the Object Database table and is the visible manifestation of the script with which you work in the script editor. Changes you make to this form of the script are saved when you save your Frontier.root file.

The compiled form of the script is actually a compact, "tokenized" format. This version of a UserTalk script is created when one of these events occurs:

- The user clicks on the "Compile" button in the script editor.
- The script is called from another script (which might be an agent, a menubar script, a Quick Script command, or even an incoming IAC message).

- Frontier is started and the script has been stored in the system.agents table.

This "split personality" of a UserTalk script has a number of implications. Two are of special interest here. First, when a script is created or loaded into memory, it has no compiled code linked to it. Second, when you close a script editing window without compiling your changes, Frontier will ask you not if you wish to save the script but if you wish your changes compiled before closing the window (see Figure 62).

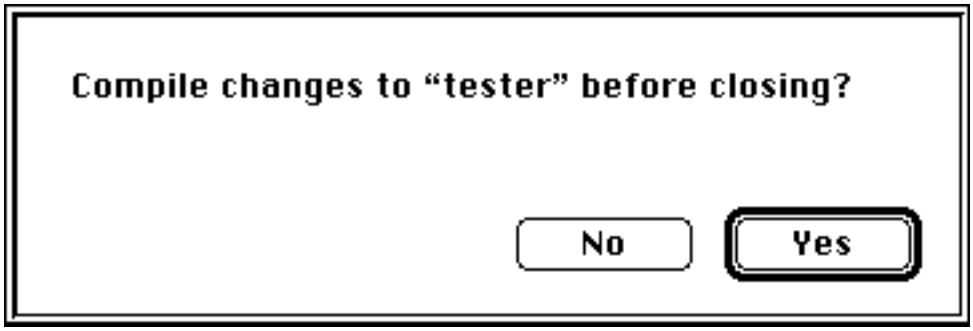


Figure 62. Dialog Confirming Desire to Automatically Compile Script Changes

Unless you are an advanced UserTalk script writer, just click on the "Yes" button in this dialog. The circumstances under which you would wish to say "No" are rare and of interest only to advanced script writers.

Contents Page | Previous Section | Next Section -- **Running a UserTalk Script**

HTML reformatting by Steven Noreyko January 1996

Chapter 6: continued

Running a UserTalk Script

There are numerous ways to execute a UserTalk script once you've created and compiled it successfully. (Actually, you don't have to compile it as a separate step; running it from its script-editing window compiles it and executes it in one step.)

To execute a script, you can:

- click on the "Run" button in the script's editing window
- click on the "Debug" button in the script's editing window and then choose a mode in which to execute the script
- type the name of the script, with parentheses enclosing parameters if required and empty parentheses if it requires no parameters, into the Quick Script window
- use the name of the script in another script, again with appropriate syntax for parameters and parentheses
- attach the script to a menu and choose the menu option

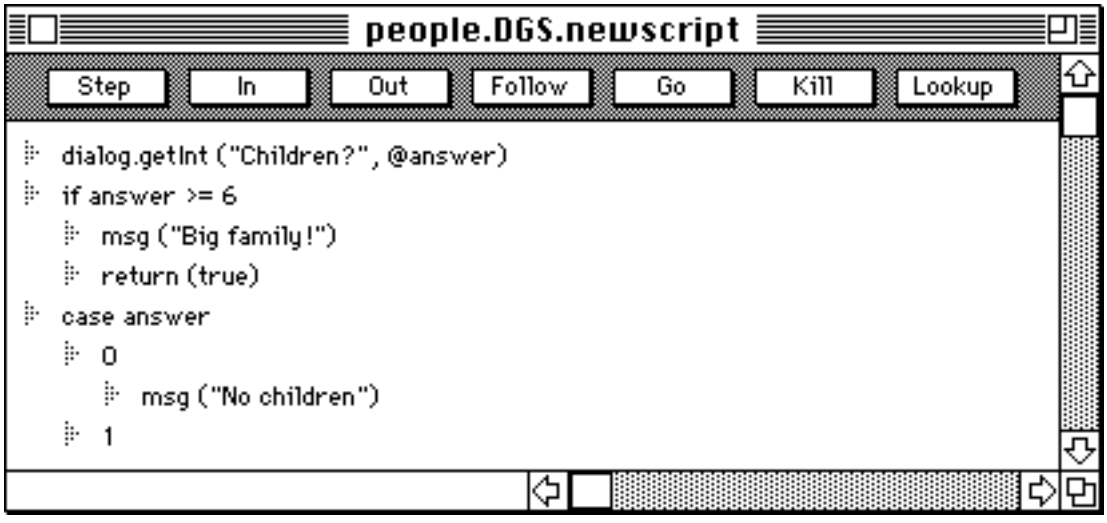
If you simply click on the "Run" button, the script will execute. Unless the script itself contains a verb that closes its window (which would be a rarity, indeed), you will be brought back to the window when the script is finished executing.

The "Debug" button, on the other hand, permits you to execute the script in any of several ways, depending on what you want to determine about the script. Figure 63 shows the buttons that appear when you click on the "Debug" button in the script editing window.

Figure 63. Script-Execution Window Buttons

From left to right, these buttons permit you to do the following things while executing a script:

- "Step" enables you to walk through the execution of the script a statement at a time. Since script execution stops at each line, you can look at entries in the Object Database, examine variables in the program and perform other debugging tasks in this mode.
- "In" allows you to "dive" into a script that is being called by the script you are debugging. Click on this button while the cursor is positioned on a line in your script that calls a local or external script. If it's an external script, Frontier will open it in an editing window and position the cursor at its first executable line. You are still in debugging mode on the called script.
- "Out" has the effect of canceling one click of the "In" button. Essentially, it tells UserTalk, "Don't stop executing again until you're out of this script and back at the level from which this script was called."
- "Follow" highlights each line of your script as it is executed but does not stop. This enables you to spot things like lines or groups that are being skipped when you thought they should be executed without having to step through the script a line at a time.
- "Go" has the same effect as the "Run" button (i.e., it executes the script to its conclusion without stopping). The difference between the two buttons is that you can click on "Go" after you've stepped through or even into and out of part of a script and just wish to complete its execution. It does not begin at the start of the script as "Run" does. You can also use "stop" to suspend execution after a "Go" command and then resume debugging.
- "Kill" terminates execution of the script immediately.
- "Lookup" permits you to examine the values of the local variables used in your script while the script is executing. Figure 64 shows an example. The script in the top window is executing in single-step mode and the user has indicated a desire to lookup the values of the variables set at this point in the script. Frontier opens an Object Database table entry where the variables are stored in a stack-like structure shown in the bottom window of the figure.



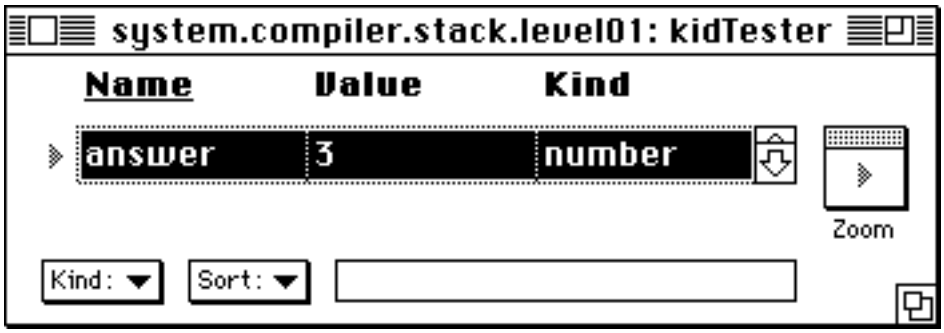


Figure 64. Sample Variable Lookup

As you can see, the variable "answer" has a value of 3 at this point. The "Lookup" button can be quite useful in debugging scripts when variables seem to be getting changed or set to unexpected values as the script executes.

We'll have more to say about how to use the facilities in the Frontier script-execution window during debugging later in this chapter.

In the same way that you can execute a script from the Frontier Quick Script window, you can also include a call to your script in another script. The approach is basically the same as in the Quick Script window: you must include a pair of parentheses whether the script requires any parameters or not. If one or more parameters are required, you should supply them between the parentheses.

Finally, you can run a script by attaching it to a menu and then choosing that menu item. Because we usually want to be able to run a script from more than one place and because a script that is attached only as a menu option is harder to access from other places in the Frontier environment, we don't generally attach the entire script to the menu. Instead, we attach a small script that calls the script we wish to execute. To see an example of this, hold down the Option key and select "Backup Frontier.Root..." from the UserLand menu. Now open its script with the "Script" button in the menubar editor. The script shown in Figure 65 appears.

Figure 65. Sample Script Attached to Menu

Notice that this script, after alerting the user to what is about to happen and confirming that it is alright to proceed, simply calls the script called backuproot in the Object Database table called backups. If the backup script itself was attached to the menu, the menu would offer the only way to execute it.

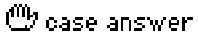
If you write a script outside the menubar editor that you wish to attach directly to a menu item, you can copy it, then open the menu item to which you wish to attach it in the menubar editor, open its script-editing window, and paste the old script into place.

Debugging Strategies

As we saw in the previous section, Frontier includes in its UserTalk script-execution window a number of capabilities that facilitate debugging your scripts. If you write relatively small scripts, you'll probably find these facilities adequate.

However, there are times when stepping through a script a line at a time is too tedious to be of much value. For such situations, Frontier includes the ability to define a breakpoint on any statement in your UserTalk scripts. When Frontier encounters a breakpoint, it stops processing unconditionally; all of the other circumstances of stepping, and moving into and out of subordinate scripts we discussed earlier are subordinate to this point.

You set and clear breakpoints by selecting the line on which you wish to change the breakpoint's status and then choosing "Toggle Breakpoint" from the Script menu or pressing the Command-K keyboard equivalent. You can also Command-click on the line's item marker. When you set a breakpoint on a line, the leader character changes from a triangular shape to a hand:



When script execution encounters this line, it stops. This works only if the script is running in debug mode, however. A script with a breakpoint will execute normally when it is invoked with the "Run" button, exactly as if the breakpoint did not exist.

You can also use other approaches to debugging UserTalk scripts, such as displaying messages in the Main Window indicating what part of a script is executing, or the value of some variable you wish to monitor without interrupting script execution.

Contents Page | Previous Section | Next Section -- **Key New Concepts in UserTalk**

HTML reformatting by Steven Noreyko January 1996

Chapter 6: continued

Key New Concepts in UserTalk

Frontier introduces several new ideas. As a scripting language, UserTalk embodies a number of evolutionary concepts such as the use of the outline as a programming model and the close integration of a database. Four of UserTalk's new ideas, though, are sufficiently unusual and useful that they merit further exploration.

These four ideas are:

- agents
- suites
- bundles
- reusable code through factoring

While it is not essential that you understand any of these notions to write effective UserTalk scripts, you will find that a familiarity with them will make you more efficient as a script writer and will enable you to deliver solutions more readily.

Agents

UserTalk supports the idea of background processes called agents. These processes are simply scripts that are stored in a specific place in the Object Database and are subject to scheduling by means of a special UserTalk verb.

Agents are stored in the table system.agents. Any script stored in this location in the Object Database will be executed repeatedly in the background (that is, while other, normal processing continues in the foreground). All agent scripts must have one line that uses the clock.sleepFor verb. (There is one exception, which we'll address shortly.) This verb is valid only in an agent script and has the effect of scheduling the script's next execution after the passage of the number of seconds provided as a parameter with the verb. If the agent is designed to execute every minute, for example, it would include the line:

```
clock.sleepFor (60)
```

When UserTalk encounters this line, it sets up a 60-second timer. When the timer expires, the script executes again.

If a particular script is to execute every second, you may omit the clock.sleepFor verb from the script. Frontier automatically executes any agent script every second in the absence of specific instructions.

Let's take a look at an agent script. In the Quick Script window, type and execute this line:

```
edit (@system.agents.MinutesSinceShip)
```

You should see a window like the one in Figure 66.

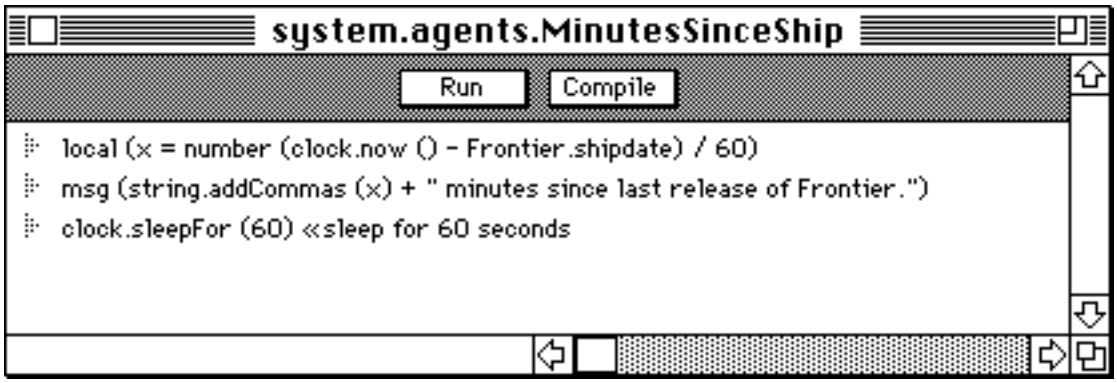


Figure 66. Sample Agent Script

Notice the last line, where the process defined in this script is paused for 60 seconds before it executes again.

Click on the "Run" button in this window. The script will execute once and then you'll see a Frontier Error Info window like the one in Figure 67.

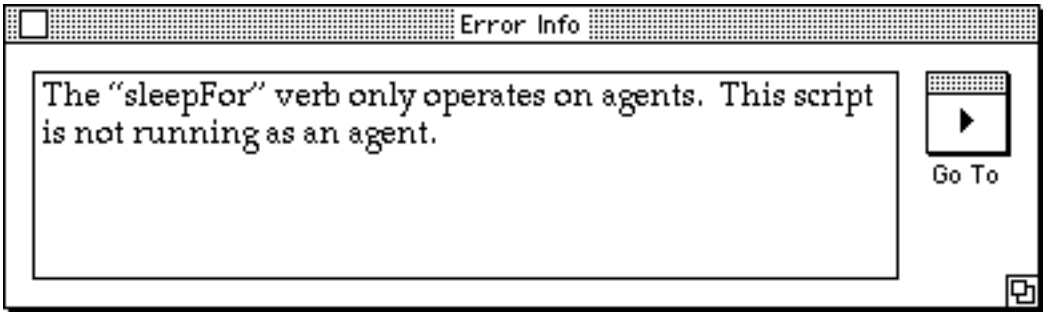


Figure 67. Error Generated by Running Agent Script from Its Window

Even though the script is stored in the proper table and includes the code to make it an agent, it is not running as an agent because you have asked it to execute in the foreground. Close the error window and the script's editing window. Now go to the Main Window and click on the down-pointing arrow. Select from the resulting popup menu the "MinutesSinceShip" option. Now you'll notice (see Figure 6-8) that the Main Window is indeed updated every minute until you suspend its ability to update the Main Window by changing back to the "StatusMessage" agent in the Main Window.

Figure 68. Main Window With minutesSinceShip Agent Running

Suites

We have mentioned suites before, but we haven't provided you with much information about how to create and use them in Frontier. This section provides you with the detail you

need.

In a sense, you can think of a suite as a Frontier application that is run from the Frontier environment. Most suites are composed of UserTalk scripts that operate on Frontier and on the system and Finder applications. A suite is a collection of one or more scripts that share a common menu-driven interface and typically carry out a set of related functions. The "Suites" menu that comes with Frontier includes a number of such suites. In Chapter 8, we'll build a suite to manage lists.

Suites have at least two things in common:

- a menubar object named "menu"
- a script called installMenu that installs the suite's menu on the menubar

Generally, suites are launched as a response to the user selecting the suite's name from the Suites menu. But this need not be the case. You could, for example, install a suite when Frontier is started or in response to the user's selection of an option from some other menu.

Suites should always redefine Command-M and Command-T to open their own menubar editors and Object Database tables.

It is perfectly legal for a suite to install more than one menu. In fact, you can define as many menus for your suite as you have room for on the menubar. When users finish using a suite, they can press Command-hyphen to remove its menu(s) from the menubar.

Aside from these differences, suites are merely collections of UserTalk scripts. A suite table can, of course, include things besides scripts and the requisite menubar: parameters and outlines with which it works, along with any other type of Frontier object it needs, may be stored there.

Bundles

A bundle is simply a way for you to take advantage of the fact that your UserTalk scripts are stored as outlines. Bundling a group of related lines together can result in code that is more readable at a higher level than the individual lines would be. It's a good idea to bundle lines that make up a multiple-line process whose details many or most readers of your script will not need to understand. Details of bundling can be found in the UserTalk Reference Guide.

There are two ways to create a bundle. The first is to type the word "bundle" into your UserTalk script and then indent the group of lines under this heading. (An easy way to indent those lines is to choose the "Demote" option from the Script menu.)The second is to position the cursor on the first line you wish to be part of the bundle and then choose "Bundle-ize" from the Frontier Script menu. You can undo a bundling operation by positioning the cursor on the heading where the "bundle" is defined or any line contained in the bundle and choosing the "De-Bundle" option from the Script menu.

Reusable Code

If you have some object-oriented or other highly structured programming experience, the idea of designing code for reusability is not new to you. In fact, you can skip this section. If

you have experience with a procedural language like Pascal or C, you probably have at least a nodding acquaintance with the idea of reusable code, but we'd encourage you to read this brief discussion anyway.

At the heart of UserTalk is a highly reusable set of verbs. You use these verbs over and over again in your scripts. You don't have to write the code for each of these verbs in every script where you want to use them. Yet, if you examine the contents of the system.verbs.builtins table, you'll see that all Frontier verbs are implemented as UserTalk scripts. Many of these scripts consist only of two lines, one defining the verb and its parameters and the second a call to the Frontier program's language kernel. But many of these verbs are completely implemented in UserTalk.

You should design your UserTalk scripts so that each script performs a single task that can be isolated. Then you may find you can re-use a script in many other UserTalk applications. For example, if you're designing a suite that always has to open a particular outline object for editing by the user, write one script whose job is to open that outline. Then in any other scripts that need to open the outline, rather than having them open the object directly, use a call to this single outline-opening script. This may seem inefficient, particularly because opening an outline for editing only takes one line of code. But it has one major advantage: if you change the location or name of the outline, you only have to change one line of UserTalk code; the other scripts will continue to work as expected.

This increased ease of maintenance is a key argument in favor of highly factored, reusable code. (By "highly factored," we mean code which has been properly divided by operation or purpose and isolated so that it is easy to find and use.) It becomes even more important when the operation you are factoring into its own script requires a number of lines to execute. By creating a reusable script for this purpose, you are not only improving maintainability, you are also increasing efficiency since these multiple lines only exist in one place and therefore need not be stored with each script that requires the functionality.

HTML reformatting by Steven Noreyko January 1996

Chapter 7

The UserTalk Language

This chapter provides you with an overview of the UserTalk scripting language as well as several language-related issues of interest to script writers. It is not a definitive reference on the UserTalk vocabulary and syntax; for that information, see the UserTalk Reference Guide.

After a brief overview of the verbs that make up the UserTalk language, we will look at the use of variables. Well show you where to find more information about the language while youre scripting in Frontier. Then we will focus on operators defined in the language and their basic usage. Finally, we will turn our attention to the main types of data with which you will work in your UserTalk scripts.

The UserTalk Verbs

The vocabulary of UserTalk consists of several hundred verbs. These verbs are identical to functions in conventional programming languages; we refer to them as verbs because they are most often used to direct Frontier or some other program with which Frontier is interacting to do something. Action is the province of verbs in human speech.

Most of the verbs in UserTalk can be divided into 23 categories. These categories do not include a number of verbs that are considered advanced and that are covered in DocServer. Some of the categories of verbs have sub-categories. In the following sections, well look at each category of UserTalk verb. Where we indicate the number of verbs of a particular type or category, you should remember that we are talking about the basic vocabulary of UserTalk. There may well be other verbs of a particular type that arent covered in this discussion or in the UserTalk Reference Guide.

Table 71 summarizes the 23 categories of verbs described in the UserTalk Reference Guide.

Table 71. Summary of UserTalk Verbs

Category	No. of Verbs	Uses, Comments and Notes
Basic	29	Dealing with numbers, datatypes, and objects. Other miscellaneous functions.
Clock	7	Setting and getting system clock values. Time-stamping documents. Pausing execution.
Date	2	Getting and setting date values
Dialog	10	Displaying and returning results from dialog boxes
EditMenu	14	Emulates behavior of Frontiers Edit Menu, including fonts and s
File	64	All file operations, including copying, deleting, moving, creat

		Also supports looking at file and folder contents, reading and dealing with volumes, parsing path names, locking and unlocking Macintosh aliases,and other related operations.
FileMenu	9	Emulate behavior of the Frontier File Menu
Finder	23	Manage and manipulate the Macintosh Finder, including windows, and folders, and the Finder as an application
Frontier	5	Turn agents on and off, make Frontier the active application, a information about Frontier
Keyboard	4	Determine if the Command, Control, Option, and/or Shift key is
Launch	5	Open applications, control panels, documents, and code resource
Main Window	6	Control Frontiers Main Window
Menu	9	Manage and manipulate the Frontier menubar and menubar objects
Mouse	2	Find out where the mouse is and if the button is being pressed
Outline Proc.	32	Edit, navigate in, rearrange, expand, collapse, retrieve inform otherwise manipulate Frontier outline objects
Category	No.	of Verbs Uses, Comments and Notes
Search	6	Handle Frontiers internal find-and-replace operations
Speaker	2	Define the sound the internal speaker will make and activate it
String	25	Manage and manipulate string objects, including word- and sente Includes system-information strings and hexadecimal value usage
System	4	Deal with external applications and find out what version of th Operating System is in use
Table	13	Manipulate and interact with Frontier Object Database tables an
Target	3	Determine the Frontier object that will receive the next action taken in the environment
Window	21	Manage and manipulate Frontier windows
Word Proc.	25	Handle word processing objects and their contents, including se controlling the selection, formatting the appearance of text an

Within each category of Frontier verbs defined in Table 7-1, there are individual verbs. The full name of a verb is composed of its category, followed by a period, followed by the name of the verb. For example, the verb that tells you the position of a Frontier objects window is window.getPosition. Because verbs behave like traditional programming language functions, each returns a result. You can use this result to nest functions inside one another. Continuing our example, if you wanted to know the position of the frontmost window, you could write:

```
window.getPosition(window.frontmost(), @horiz, @vert))
```

The first argument needed by window.getPosition is the title of the window whose position you want to know. The verb window.frontmost, which takes no arguments, returns a string containing the title of the frontmost window. As you can see, any UserTalk verb that does not require any parameters must still include its set of parentheses.

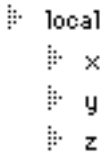
The UserTalk Reference Guide contains a complete description of each of the 320 verbs in Table 71, including a discussion of their parameters, the values they return, notes about their usage, and examples. Essentially the same information is available on-line via the DocServer application.

Variables in UserTalk

As with most programming or scripting languages, UserTalk supports two types of

variables: global and local. Global variables are those that are permanently stored in the Frontier Object Database and have cell addresses there. Local variables are those that are used temporarily within a script or local script and exist only within the context of those scripts.

You should get into the habit of declaring local variables, particularly in scripts that can be called from other UserTalk scripts. To define a local variable, use the keyword local. When it is used to declare one local variable, its usage generally looks like this:




```
local (x = 3)
```

The assignment of an initial value is optional. It is perfectly permissible to declare the same local variable as:

```
local (x)
```

When you wish to declare more than one local variable, you can use the keyword in one of two formats. The first is identical to that for single variables, except it is repeated:

```
local (x, y)
```

You can initialize variables in this form as well:

```
local (x = 3, y = 4)
```

The alternative is to use UserTalks indentation to cut down on typing (and, incidentally, to allow the declarations to be collapsed see listing on below).

On-Line Documentation

While you are scripting in Frontier, you can look up information about specific UserTalk verbs and other language elements on-line. You may find yourself needing to go to the printed UserTalk Reference Guide relatively infrequently compared to other programming and scripting languages you may have used. There are two places where information about the UserTalk language resides on-line: in Frontier itself and in a special application provided with Frontier called DocServer.

Contents Page | Next Section -- **References in Frontier**

HTML reformatting by Steven Noreyko January 1996

Chapter 7: continued

References in Frontier

As we noted in Chapter 4, the UserLand menu in Frontier contains a hierarchical sub-menu dedicated to on-line documentation. These on-line documents can be broken down as follows:

- index that covers a number of subjects
- database map suite that enables you to find things in the Object Database
- list of datatypes
- list of verbs by category
- complete formal grammar of UserTalk

The index resembles the index you might expect to find in a printed manual, with the exception of the fact that you can Command-double-click on its actual index entries and be taken to the place in the Object Database where the object involved is stored. Figure 71 shows a small part of this index. By Command-double-clicking on the last line shown in the Figure, you ask Frontier to open a document explaining how to install changes to UserTalk scripts.

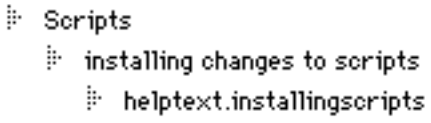


Figure 71. Segment of On-Line UserTalk Index

Selecting the Object Database Map option from the UserLand menu will install a new menu on the Frontier menubar. This menu is labeled Map and permits you to open the current version of an outline stored at suites.tablemap.thelist or to rebuild it so that it reflects all of your latest changes to the database structure. It also permits you to position the cursor on any line in the outline and press Command-J or select Jump Through from the Map menu to be taken to the object editing window of the object on which the cursor is positioned.

If you choose the Frontier Data Types option from the UserLand menu, Frontier will open the outline helptext.datatypes. This outline, a portion of which is shown in Figure 72, lists all of the valid UserTalk datatypes. We will have more to say about datatypes later in this chapter.

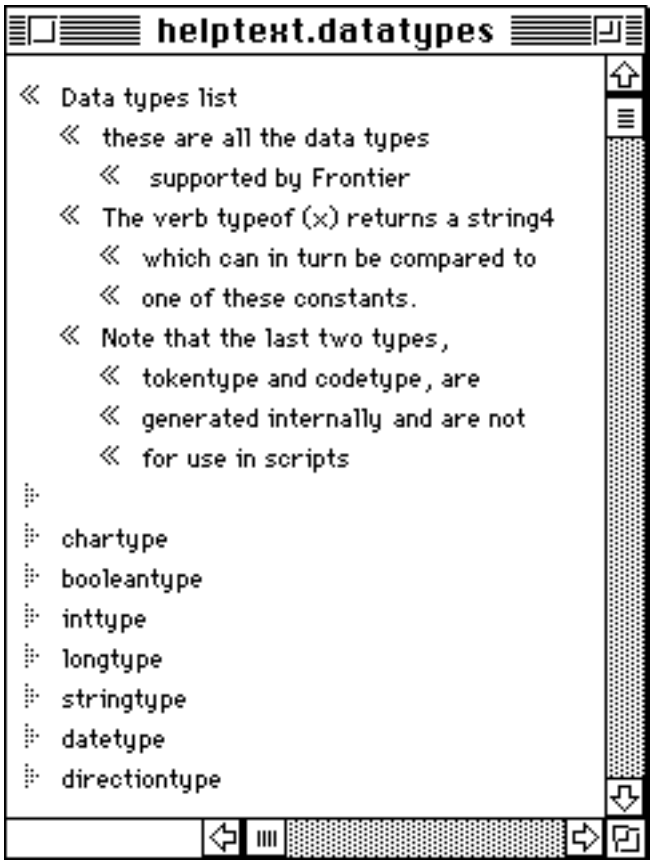


Figure 72. Partial Listing of Frontier Datatypes List

You can open the suites.doc.verblist outline by choosing the Frontier Verbs option on the UserLand menu. This outline lists all built-in UserTalk verbs, including those that are covered in DocServer. Figure 73 shows part of this outline.

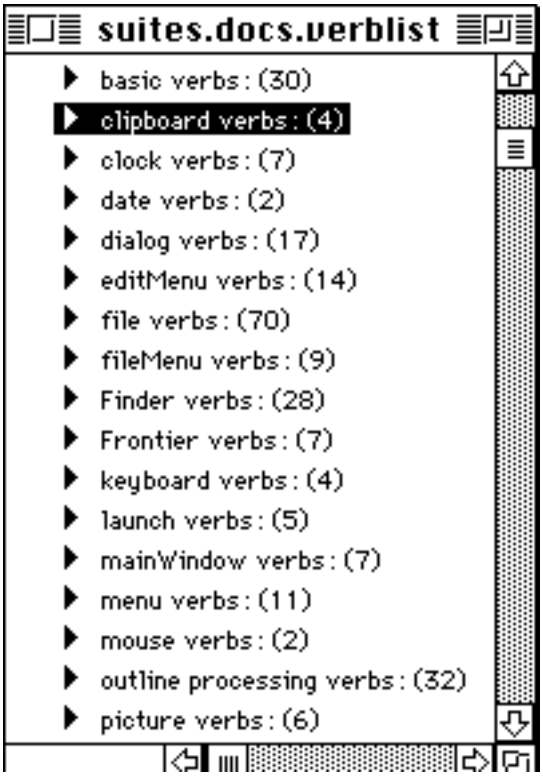


Figure 73. Part of UserTalk Verb List

The formal grammar of UserTalk is available via the Frontier Grammar entry on the UserLand menu. Selecting that option opens an outline called helptext.grammar, part of which is shown in Figure 74. The outline contains a formal grammatical description of the language, using standard symbols in the computer science profession. If you have trouble understanding it, thats probably a good sign you dont need to worry about understanding it. A copy of this outline can also be found in Appendix B to the UserTalk Reference Guide.

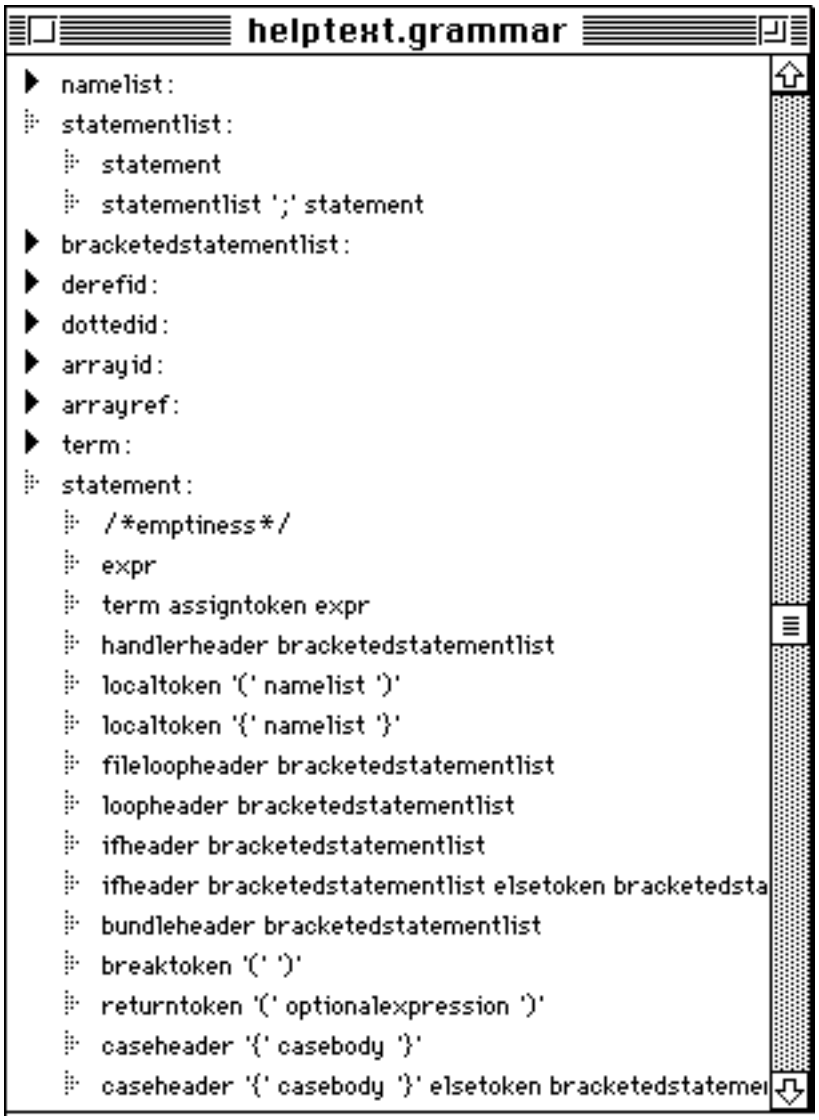


Figure 74. Part of UserTalks Grammar Window

Contents Page | Previous Section | Next Section -- **DocServer Documentation**

HTML reformatting by Steven Noreyko January 1996

Chapter 7: continued

DocServer Documentation

Frontier is shipped with a product called DocServer. This is a tightly integrated, IAC-aware application that gives you a fast and easy way to access information about the 320 verbs that make up the core of the UserTalk scripting language as well as all of the advanced verbs. You may wish to keep DocServer running any time you are scripting in Frontier.

The application is found in the UserLand Utilities Folder on your Frontier disks.

There are two ways to access information in the DocServers database of information about UserTalk verbs.

First, you can simply run DocServer and use its menus (discussed below) to move around in the on-line document.

Second, you can hold down the Control key and double-click on the name of any verb in the Frontier environment in a script, in the Quick Script window or in an outline, for example and DocServer will move to that page. In fact, if DocServer isnt running when you Control-double-click on a verb, Frontier will launch it and then turn to the page on which the verb you are interested in is discussed.

Once you are in DocServer, you can use its menu to move to the first or last verb in the on-line document or to look for a specific verb. For the latter, press Command-J or select Jump To Verb... from the DocServer menu. You will be asked what verb you want to find. Once you answer, DocServer will turn to that verb in the documentation.

The Frontier menu in DocServer permits you to launch Frontier, open the special verb list associated with DocServer or build a new verb list for DocServer.

NOTE

Both the DocServer menu and the Frontier menu are implemented using Frontiers Menu-Sharing protocol. This protocol is designed for people creating IAC-aware applications to work with Frontier. It is an integral part of the Frontier Software Developers Kit (SDK), which is included with Frontier.

UserTalks Operators

Table 72 summarizes the operators that UserTalk understands. See the UserTalk Reference Guide for details.

Table 72. UserTalk Operators

```
Operator    Purpose
  +   Addition of numeric values or concatenation of string and character values
  -   Subtraction of numeric values or sub-string extraction of string values
  *   Multiplication
  /   Division
  %   Modulus
  = = equals  Test for equality
  !=  notequals   Test for inequality
  <   lessthan    Less-than comparison
  < = Less-than-or-equal comparison
  >   greaterthan Greater-than comparison
  > = Greater-than-or-equal comparison
  ||  or  Logical OR
  &&  and Logical AND
  !   not Logical NOT
  ++  Increment
  - - Decrement
  @   Address-of
  =   Assignment
  ^   Dereference
```

These operators are common to most modern programming languages; their use in UserTalk is not substantially different from what youd expect if you have other programming or scripting experience. The address-of operator (@), however, requires further explanation.

You will use the address-of operator in conjunction with a UserTalk verb that requires an address as an argument. This operator returns an address, rather than the value stored at that address in the Object Database. Here is a brief example. In the examples table, there is a numeric value stored at the location called age. If you wanted to add a number to that value, you would use a line like this:

```
16 + examples.age
```

Frontier would obligingly return the answer.

If, on the other hand, you type:

```
16 + @examples.age
```

Frontier will oblige you with an error message. Thats because youve tried to add a number to an address. UserTalk tries to convert the number 16 to an address, fails, and lets you know its unhappy.

Where the UserTalk Reference Guide indicates that an address is required, you should precede the Object Database location of the object with the @ operator. It is often permissible to use a string rather than the address, and Frontier will attempt to coerce the string to an address value if it needs one, but it is much more straightforward simply to define the address as an address in the first place.

UserTalk Datatypes

UserTalk is a rich language that is capable of dealing with no less than 28 different types of data. Many of these datatypes are useful only to advanced script writers and are discussed in DocServer. The primary datatypes with which you will be likely to work are summarized in Table 73.

Table 73. Common UserTalk Datatypes

Datatype	Range of Legal Values	Notes
addressType	Any Object Database cell or any non-existent cell in an existing table	
booleanType	True, False, 1, or 0	1=True, 0=False
charType	Any ASCII character (0-255)	Enclosed in single quotation marks
dateType	Any legal system date value	
directionType	up, down, left, right, flatup, flatdown, nodirection	
intType	-32768 to 32767	
longType	-2147483648 to 2147483647	
menubarType	Any Frontier menubar object	
outlineType	Any Frontier outline object	
scriptType	Any legal UserTalk script	
stringType	One or more legal characters Enclosed in double-quotation marks	
string4Type	Any four characters (exactly) enclosed in single quotation marks	Used by the Macintosh Operating Syste
tableType	Any Frontier Object Database table	
wptextType	Any Frontier word processing text object	

HTML reformatting by Steven Noreyko January 1996

Chapter 8

Scripting the Frontier Environment

This chapter discusses how to write UserTalk scripts that operate entirely within the Frontier environment. It is the first of three chapters that deal with specific examples of how to use UserTalk to develop real-world utilities and applications. The next two chapters focus, respectively, on scripting interaction with the Macintosh System and Finder, and on scripting for interapplication communication (IAC).

We will begin this discussion with an overview of scripting in the Frontier world, emphasizing the kinds of things you might want to script in UserTalk. Then we'll provide three concrete examples of UserTalk scripts that function entirely within Frontier. The first is a small demonstration of how you can add a capability to the Frontier menubar with a minimal UserTalk script. The second is an activity tracker that you will find quite adaptable to handling all kinds of lists associated with dates. The last example is an agent (background process) that you can use to remind yourself of appointments. All of these examples are deliberately simple; our objective is not to build commercially robust software but to demonstrate scripting techniques and approaches. You will undoubtedly have dozens of ideas for ways of extending and enhancing these scripts. View them as starting points.

UserTalk in Frontier: An Overview

Frontier is an application, but like all applications of the future it is also an application development platform. Containing as it does an Object Database, a customizable menubar, and a robust scripting language, it is a rich environment in which to build even fairly sophisticated applications that require minimal user interfaces. Programs that need more robust interfaces are best developed outside Frontier, in a programming environment like THINK C, and "wired" to Frontier as described in Chapter 10.

Applications you write in UserTalk for use entirely within Frontier will almost certainly involve manipulation of the database. As you know, the very act of scripting alters the Object Database because it is there that your scripts are stored. Beyond that, though, manipulating the database - altering its contents, changing program direction based on contents, displaying information stored in it - will play a central role in most applications you will build in UserTalk if their intended environment is Frontier.

You should be familiar with how the Object Database is organized and used. This would be a good time to review Chapter 5 of this manual if you are unclear on any of these issues.

The menubar is part of the Object Database. As we have seen, this menubar can be customized in any of several ways. One of our examples will focus primarily on this aspect of UserTalk scripting for Frontier-contained applications.

First Example: Customizing Menus

People who work with outlines a lot tend to adopt a certain style for approaching information. They are able to see information in discrete chunks related to specific topics and sub-topics. Often, they wish to view only a part of an outline. To facilitate this, many outliners, including the Frontier version, support the notion of hoisting headlines. A hoisted headline acts like an outline within the outline. When you hoist a headline, its sub-headings appear at the left margin of the outline-based window in which it is displayed.

Although Frontier supports hoisting and its counterpart, de-hoisting, it does not include on its menubar commands to handle these processes. We'll add those capabilities in this first example.

The Steps

As with any Frontier scripting task that involves menu modification, this example involves the following steps:

1. Identify the menu on which you wish to make the change.
2. Map out the way you want the menu to appear when you're finished. (You need not do this, of course, if the change is trivial.)
3. Make the changes to the menu in the Frontier menubar editor.
4. Connect appropriate scripts to the new menu entries.
5. Test the result.
6. Save the updated root.

Identifying the Menu

The first step is easy in our example. We want to be able to hoist and de-hoist headings in an outline, so we want to change the Outline menu. If you find these new capabilities useful, you might also wish to add them to the Menubar and Script menus.

Mapping the New Menu

The second step is only slightly more complex than the first in our example. You can put the two new options anywhere on the Outline menu you like, particularly if you're the only person who will be using this enhancement to Frontier. But if you have some experience with outline processors, you might find that you'll feel comfortable with these new commands being placed either in a separate category with a menu separator between them and other groups of functions or combined with the Promote and Demote options. We chose the latter because that's how we tend to think of this type of outline processing.

One of the nice features of Frontier, of course, is that if you choose to put these new menu options someplace and a user decides to put them elsewhere, it's easy to move them.

Open the Outline menubar. The easiest way to do this is probably to open an outline and then hold down the Option key while you select any item in the menu. You may also wish to use the "Jump To" option from the UserLand menu or its Command-J keyboard equivalent. The effect will be the same in either case. You may have to move the bar cursor. Ultimately, you want to get to the position shown in Figure 8-1.

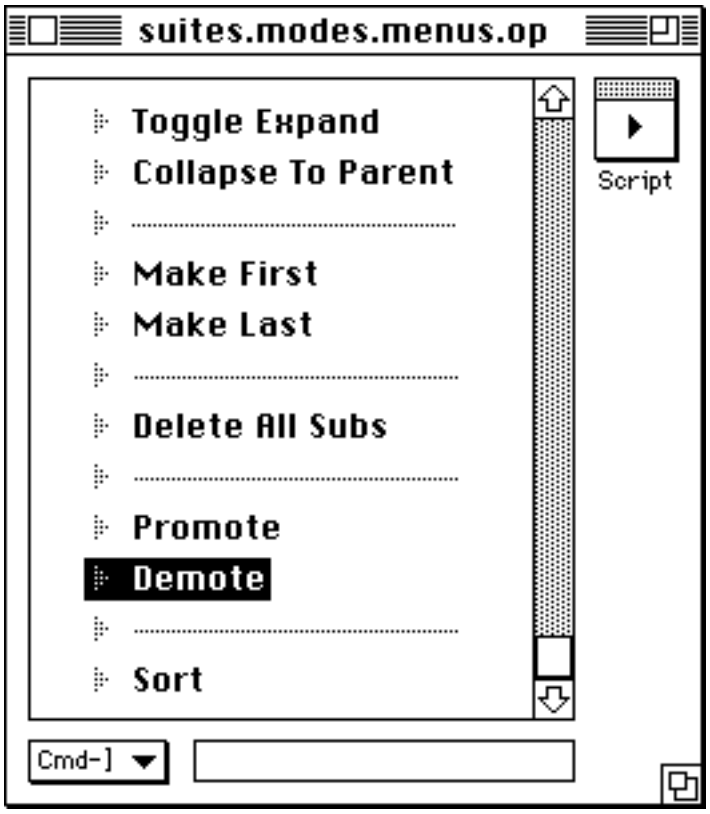


Figure 8-1. Outline Menu in Menubar Editor

Changing the Menu

Just press the Return key and type the word "Hoist." Then press Return again and type "De-Hoist." (If you wish to add Command-key equivalents for these operations, you can do so now by selecting one from the "Cmd:" popup menu in the lower left corner of the window.)

This portion of the menu should now look like Figure 8-2.

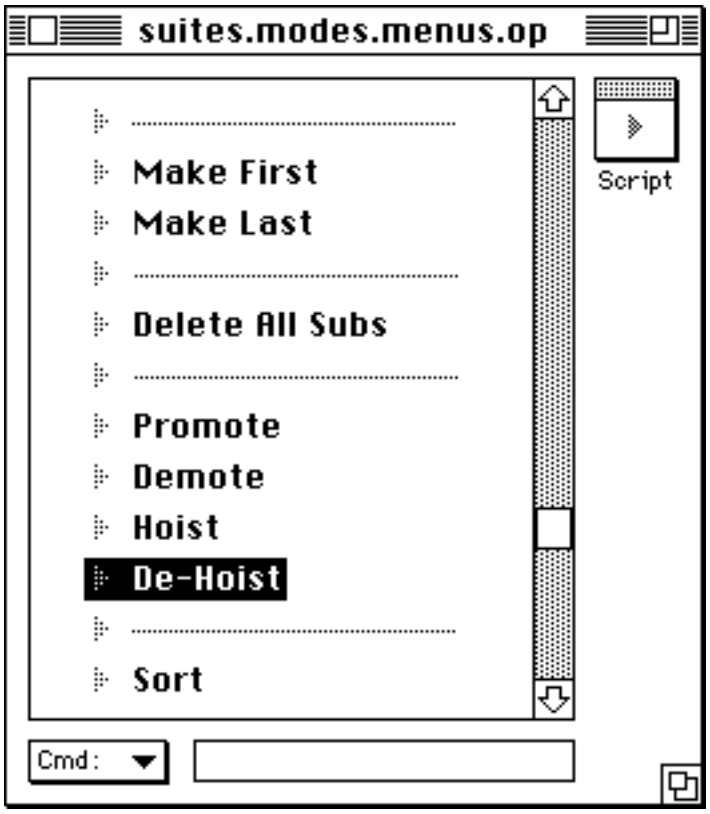


Figure 8-2. Revised Outline Menu

Contents Page | Next Section -- **Connecting the Scripts**

HTML reformatting by Steven Noreyko January 1996

Chapter 8: continued

Connecting the Scripts

The scripts to carry out the hoist and de-hoist operations can be simple because each involves the use of only one UserTalk verb. The verb that supports hoisting in an outline is called op.hoist. Its counterpart is called, logically enough, op.deHoist. (These verbs, and the rest of the UserTalk language, are documented in the UserTalk Reference Guide.)

Select the "Hoist" item in the Outline menu and click on the "Script" button in the upper right corner. Your window will look something like Figure 8-3.

Type the following one-line script in this window:

```
op.hoist()
```

Now close the window. Select the "De-Hoist" entry in the menu, click on the script button and type this one-line script into the resulting window:

```
op.deHoist()
```

Close the second script window and the menubar editing window.

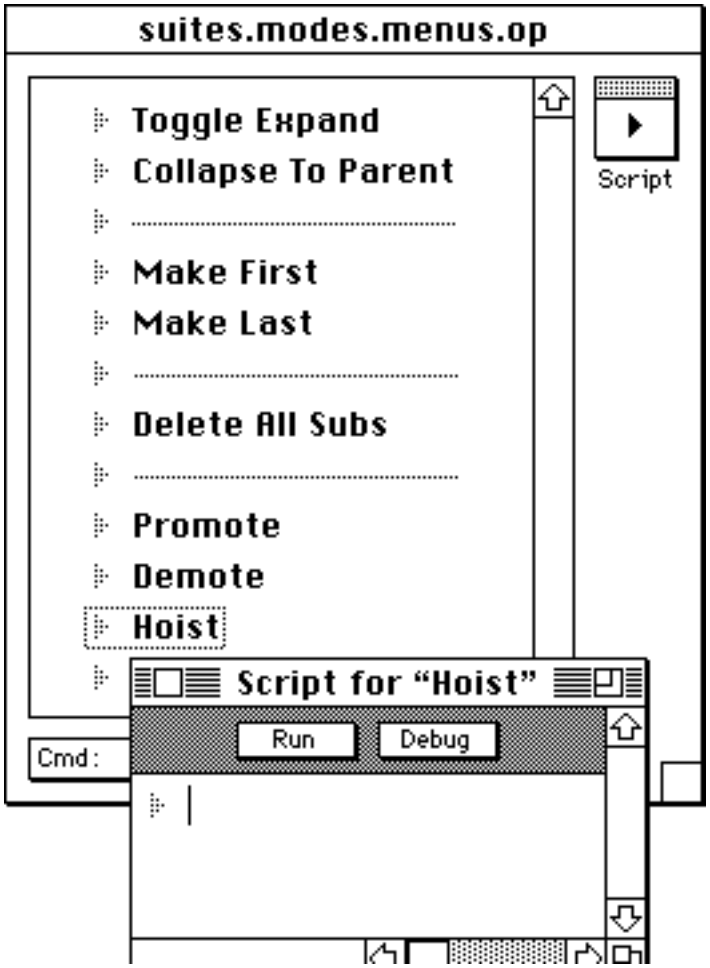


Figure 8-3. Script Editing Window Open in Menubar Editor

That's all there is to these changes. We should point out, however, that it is usually one step more complex to add functionality to a menu. Normally, the script you want to attach to a menu item is more complex than a single-line script that simply calls a built-in UserTalk verb. In such cases, you will generally store the script you want to call in some other Object Database cell. You will then call that verb from the menu. For example, let's assume that you want to call a script called hoistedPetard from an Outline menu item. Further, let's assume you've stored this script in your people table (a practice we generally discourage because it makes your scripts less general and re-usable). The script for that would look like this:

```
people.ME.hoistedPetard ()
```

In other words, you'd have to define the script to be called and then the one-line script to call it in the menubar.

There is no reason, by the way, why a script called by a menu item can't be arbitrarily complex. Most of the time, though, you will keep these complex scripts outside the menubar portion of the Object Database because they're more directly accessible and are easier to re-use in other scripts.

Testing the Scripts

Let's see if what we did worked. In the Quick Script window, type and execute this line:

```
edit(@examples.["Sample Outline "])
```

This will open a sample outline as shown in Figure 8-4.

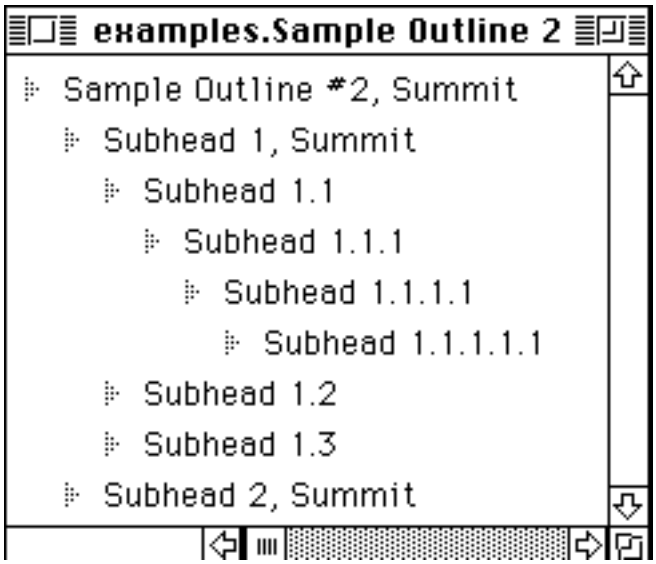


Figure 8-4. Sample Outline Open for Editing

Select the heading "Subhead 1.1" and then choose the "Hoist" option from the Outline menu. The result should look like Figure 8-5.

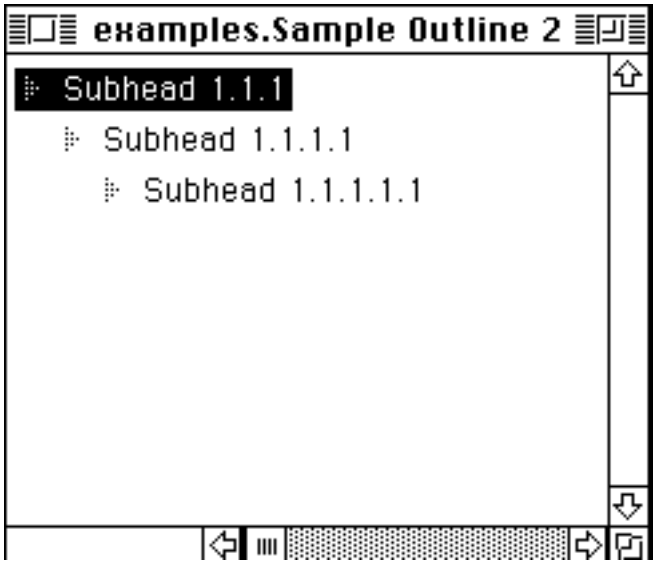


Figure 8-5. Subhead Hoisted in Sample Outline

Now select the "De-Hoist" option. The outline will return to its original view as shown in Figure 8-4.

Save the Updated Root

If you want to keep these changes to your Outline menu, you should now save your root. You can do this by selecting "Save" from the File menu or by typing Command-S. This

ensures that the next time you load and run Frontier and open an outline document, you'll have the Hoist and De-Hoist options available on the Outline menu.

Contents Page | Previous Section | Next Section -- **Second Example: Activity Tracker Suite**

HTML reformatting by Steven Noreyko January 1996

Chapter 8: continued

Second Example: Activity Tracker Suite

The second example Frontier application we'll build is an activity recorder. You can use this to keep track of all the things you accomplish each day so that on those days that you feel like you're just spinning your wheels you have a nice, uplifting place to see all that you've done recently.

There are many different approaches to designing and building applications, of course. We aren't interested in programming methodologies here, so we'll just follow a relatively simple approach to designing and building this program. It consists of the following steps:

1. Describe the desired functionality.
2. Define the needed UserTalk scripts.
3. Define and build the menu to call these scripts.
4. Write the scripts.
5. Attach the new scripts to the new menu.
6. Create any empty documents or objects needed by the scripts.
7. Test the mini-application.
8. Add the mini-application to the Suites or Custom menu.
9. Test the launch of the mini-application from a menu.
10. Save the revised root.

Describing the Desired Functionality

The mini-application we want to build will allow us to keep a running chronological list of our accomplishments. Each day will be a summit in a Frontier outline. Under each day will be one or more sub-headings, each containing a brief description of some task we accomplished that day. The outline will look something like Figure 8-6.

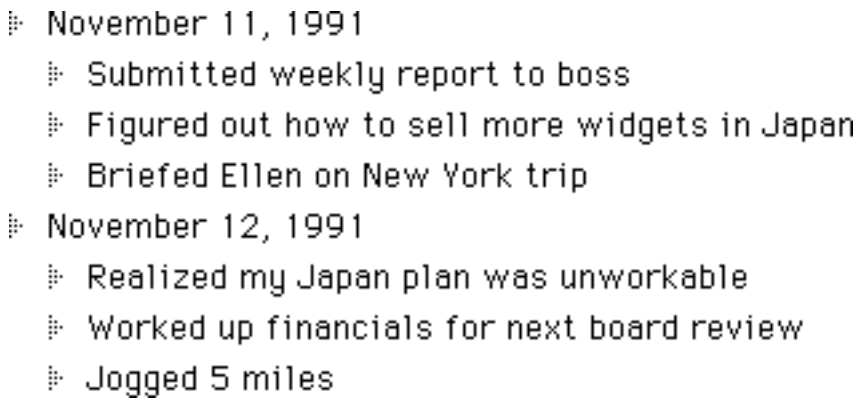


Figure 8-6. Sample List of Accomplishments

We want to be able to add items to the list as simply as possible. The program should keep track of today's date and make sure that it adds things to the right day's list. We also want to be able to ask the program how many things we've accomplished since we started keeping the list.

Notice that because we are building this application in Frontier, which has native support for outlines, there are a number of things we don't have to build into the program. For example, printing is handled for us. So are deleting items, editing their contents, and examining them in context. Taking advantage of the outline capability in Frontier is one way we can see that Frontier is a real application development platform in addition to being an application in its own right.

Defining the Needed Scripts

From the description of what we want our mini-application to do, we can come up with a list of the scripts we'll have to write.

We need a script we can invoke with a menu command that asks us to identify an accomplishment, opens the list, finds out if this is the first one for today, and place appropriately, adding a heading for today's date if necessary. We also need a script that will tell us how many items we've added to the list since we created it.

Implied but not explicitly described is a need for another script: one that will simply open the list so we can review it, print it, or work with it.

Because this mini-application involves several menu choices,we'll make it a Frontier suite. That means we also have to have three other scripts: one to edit this suite's menu, one to edit its table, and one to install its menu when the mini-application is launched by the user. Those are requirements of all suites in Frontier.

So we have identified six scripts we need to write.

Defining and Building the Menu

Since this is a suite, it will have its own menu. Frontier does not insist that this mini-application's menu and other information reside in a particular place, but we suggest you place it in the suites table. This means we have to add a new table to the suites table in which we can store this mini-application's menu, scripts, outline, and other data. Let's call this new entry "Activities."

Open the suites table by typing and executing the following line in the Quick Script window:

```
edit (@suites)
```

Now press Command-Return to create a new entry and type the name "Activities" into the Name column. Select "Table" from the "Kind:" popup menu. Double-click this new entry's item marker and you'll find yourself editing an empty table with an appropriate name (see Figure 8-7).

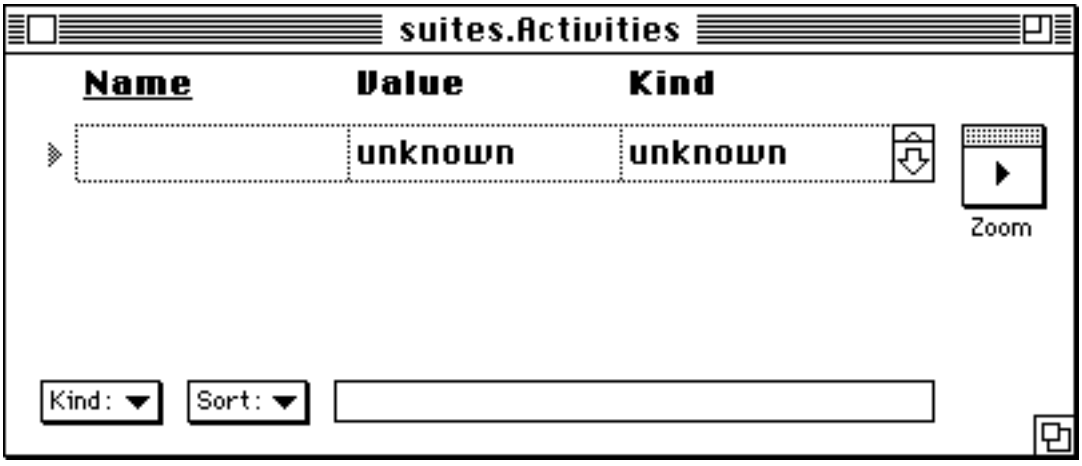


Figure 8-7. Empty "Activities" Table

Since the first thing we want to build is this suite's menu, let's enter it into the table. Type "menu" and then pick MenuBar from the "Kind:" popup. (You could rather simply double-click on the item marker next to the new table entry to produce a dialog from which you can choose the object's type.) Double-click the new table entry's item marker and you'll be in a menubar editor (see Figure 8-8).

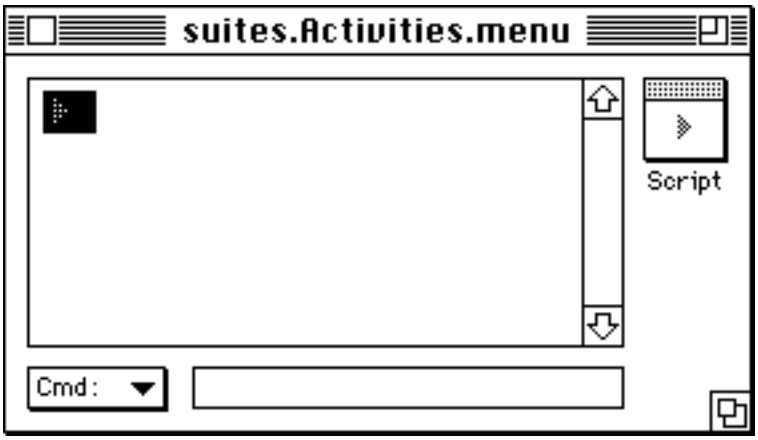


Figure 8-8. Blank Menubar Editor

Notice the window's title. You are editing the menu entry in the Activities sub-table in the suites table at the root level of Frontier.

A menu has one summit, which is the name that will appear on the menu bar when the menu is displayed. Call it what you like; we'll use "Tracker," but you could easily use some other name if you prefer. Type this title in at the summit of this menubar outline editor. Press Return and then Tab.

The first item in this menu ought to be the one we expect to use most often. That's probably adding items to the list, so we'll put it first. Type "Add an Item" but before you press Return, pause for a moment. If we're going to be really busy adding things to this list of accomplishments, we might want a keyboard shortcut for invoking the process. Since it's so easy to add this feature in Frontier, let's do that. Deciding which character to use is a bit tricky. You might be tempted to use "A" since it's the first letter of the menu entry. The problem is, the Edit menu uses this Command-key equivalent to select all of the text. You

might well want to be able to do that with the Tracker mini-application open. But the way Frontier handles Command-key processing, it looks from right to left and top to bottom for the keystroke being pressed. If you put Command-A out this far to the right, the Select All option on the Edit menu won't be accessible from the keyboard. We chose to use Command-I ("Item") instead. If you prefer some other key, feel free to substitute it. Don't use Command-M or Command-T, though, because we're going to define those later in this menu.

Look at the Frontier menubar. Notice that it is building your menu as you create it. This enables you to check it and make sure it looks like you want it to look before you quit working on it.

Following the same procedure, add the following items to the Tracker menu:

- Edit the List
- How Many Activities?
- Edit Menu (Command-M)
- Edit Table (Command-T)

The menu as you view it in the menubar editor should now look like Figure 8-9. Its drop-down version looks like Figure 8-10.

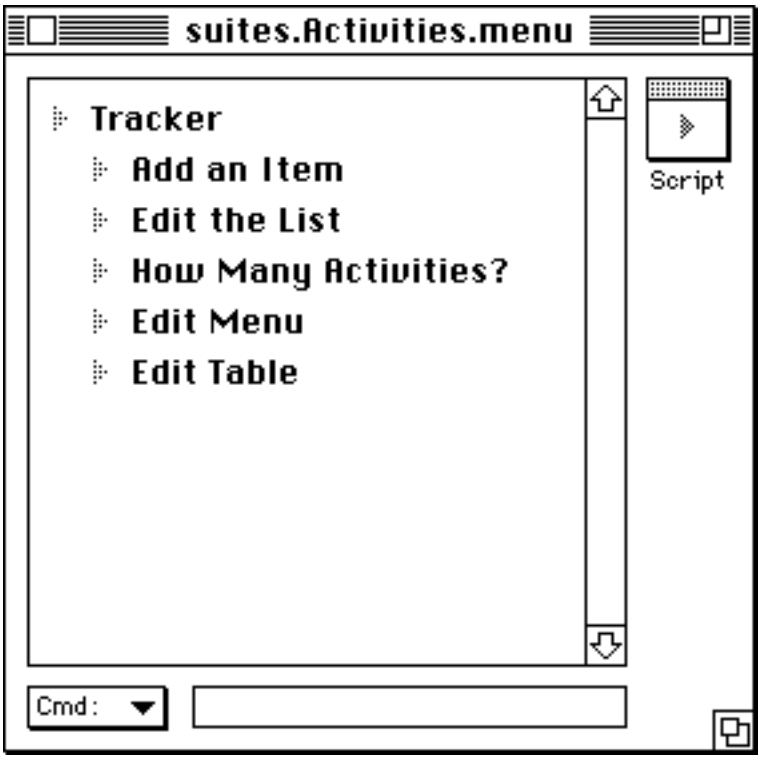


Figure 8-9. Tracker Menu Nearly Complete in Menubar Editor

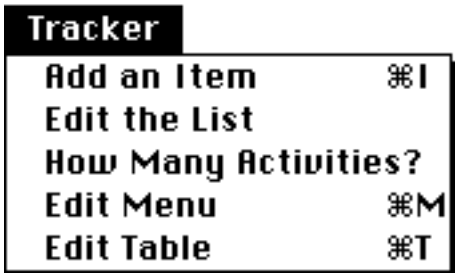


Figure 8-10. Tracker Menu Nearly Complete on Menu Bar

There is one thing wrong with the menu as it now stands. All of its functions are jammed together. There aren't any separators (dashed horizontal lines) to separate them into logical groups. Let's add two separators, one after "Edit the List" and one after "How Many Activities?". To do that, just position the bar cursor in the menubar editor on the line below which you wish to insert a separator (in the first case, "Edit the List") and press Return. Now type a single hyphen. Repeat the process for the second separator. As long as we're here, let's make another change. Select the text "Add an Item" and type an ellipsis (Option-semicolon) at the end of it. This lets the user know that when this menu item is selected, a dialog box will appear in which the user will be expected to provide some information. Now the menu looks like Figure 8-11 when it is opened.

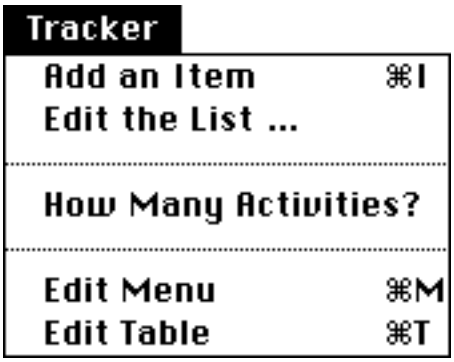


Figure 8-11. Final Version of Tracker Menu Pull-Down

As you can see, this is a much cleaner and easier-to-read menu.

Contents Page | Previous Section | Next Section -- **Write the Needed Scripts**

HTML reformatting by Steven Noreyko January 1996

Chapter 8: continued

Write the Needed Scripts

We need six scripts, as we've already determined. It doesn't matter what order we write them in (except that in a larger suite we might find that the order would help us to test parts of the project before we'd finished all of it). Let's start with the easy ones. (Actually, in "real life," sometimes it's better to start with the hard ones!)

To write the script that will open the list of accomplishments for editing, we need to know at least what we want to call the outline. If you're interested in incrementally testing things, you'll have to create this outline now but we're going to forego that nicety in the interest of keeping similar things together in our explanation. We know we're going to store the outline in the Activities suite table. Let's call it "Accomplishments." So we're going to write a script to open that outline.

Create a new entry in the suites.Activities table. Name it editList. Identify it as a script object and open it by double-clicking its item marker. This is a simple one-line script:

```
edit(@Activities.Accomplishments)
```

(Because suites is a known path in Frontier, we don't need to include the full name of the table, only enough to enable Frontier to locate it. See Chapter 5 for a discussion of paths in Frontier.)

Once you've entered this script, click on the "Compile" button to be sure there aren't any errors. Then you can close the script-editing window.

Let's take care of our other small scripts while we're at it. The three that we're required to supply as a Frontier suite are easy. The one that installs the suite's menu must be named installMenu. Create a new script with that name and enter this code into it:

This is the normally preferred format for a UserTalk script outside the menubar. Notice that it begins with the keyword "on" and that it is written as if it were a function (which it technically is) with empty parentheses indicating no parameter is required. The single script statement uses the Frontier verb menu.install to install the menubar stored in the Object Database at Activities.menu. It returns the result to the calling routine.

The other two menu items we are required by Frontier to supply can be handled with one-line scripts attached directly to the menu items. Open the "Edit Menu" item's script and type this line into it:

```
edit(@Activities.menu)
```

Similarly, open the "Edit Table" item's script and type these lines:

```
edit(@Activities)
```

Now let's work on the script that will tell us how many items have been added to the list of our accomplishments since we opened it. This takes a bit of thought. There are at least two possible approaches we could take. We could write a script that would open the list each time it was asked for this information and count the number of items in each heading, accumulating the total. Or we can set up a variable in our Activities table that we simply increment each time we add an item to the list. This latter will obviously be much faster to execute; its only drawback is that if the user deletes items from the list or adds them manually (outside the Tracker suite), our figure won't be right. For our present purpose, we've decided this is not an important issue.

We decide, then, to store the number of accomplishments in the table in a numeric entry called howMany. Again, if we wanted to test this function incrementally, we'd have to create this entry now. But let's just write the script as if the entry existed; we'll create it when the time comes.

How can we display information to the user in Frontier? There are a number of options in the set of verbs related to the dialog category. A quick perusal of the UserTalk Reference Guide chapter on dialog verbs results in a decision to use dialog.notify. So let's create a new script entry in the Activities table called tellHowMany. It should look like this:

```
on tellHowMany()
    dialog.notify("You've added " + Activities.howMany + " activities to your log!")
```

This is a pretty straight-forward script. The only thing you might not have seen before is the use of the plus sign as a string concatenation operator. It allows us here to place the numeric value stored at Activities.howManyÊinto the string that we display to the user in response to his request.

We're down to our final script, the one that does all the hard work in our suite. Open a new script entry in the Activities table. Call it addAccomplishment. Here is the UserTalk code that goes into this script; it is explained below.

```
on addAccomplishment ()
    local (temp = "")
    target.set (@Activities.Accomplishments) «No need to open the list.
    if not dialog.ask ("What did you accomplish?", @temp)
        return (false) «User cancelled the dialog
    op.firstSummit () «Move outline cursor to first summit in list.
    op.go (down, infinity) «Move to last summit in list
    if op.getLineText () notequals string.dateString ()
        op.insert (string.dateString (), down) «Create new summit with today's date.
    op.insert (temp, right) «Put accomplishment in place.
    Activities.howMany++ «Increment number of activities stored
    return (true)
```

This script begins in the usual way with the keyword "on." The first line declares a local variable called "temp." (Declaring variables to be local is optional in Frontier but strongly recommended; see Chapter 7 for a detailed discussion of this topic.) We give this local variable an initial value of an empty string; otherwise, Frontier will assume it is a number and give it a value of 0, which will look strange when we display it to the user in a dialog.

The second line sets the target of the following verb operations. The target is an important notion in UserTalk; it is described in detail in the UserTalk Reference Guide. Essentially, you can think of the target as the object to which all subsequent verb actions will be applied. You could also choose to open the window of the outline involved here by using the edit verb, but the target.set approach has the advantage that the user never sees the outline open, but its contents are updated appropriately.

The next two lines use another dialog verb to ask the user to describe an accomplishment. We use the "if not" approach because if the user presses the "Cancel" button when the dialog is displayed, Frontier will return a result of false. We can use that to terminate the operation since the user has not indicated a desire to add any accomplishments to the list. We store the user's response in temp.

The following five lines update the outline. First, we place the cursor on the first summit in the outline. Then we use the op.go verb to move all the way to the last summit. (Note that if there is only one summit, no movement takes place and the verb returns a value of false. Normally, we'd check for that condition, but since our only intent here is to make sure we're on the last summit in the outline, this situation causes no harm.)

The next line uses op.getLineText to read the last summit in the outline and checks to be sure that it is not the same as today's date. If it isn't, that means this is the first entry for today and a new summit with today's date needs to be created; that is accomplished by the first use of the op.insert verb. In either case, we now know we are positioned on today's date, either because it was already the last summit or because we just created it. So we insert the accomplishment into the outline.

We increment the value of the counter that's keeping track of how many accomplishments we've added. Finally, we return a value of true indicating successful completion. All UserTalk scripts should return a value of true at their successful conclusion.

Attaching Scripts to the Menu

Now that the scripts are written, we could test them before we add them to the menu. But since in this application we have only one script of any complexity and since it's only called from one menu item, we're fairly safe reversing that order.

If the Tracker menu is still on the menu bar, as it might well be, type Command-M to edit its menu. Otherwise, open the menubar in one of the other usual ways.

Select the item "How Many Activities?" This activity is the province of the script tellHowMany, so this menu item's script simply calls that routine:

```
Activities.tellHowMany()
```

Similarly, open the "Edit the List" item's script and type:

```
Activities.editList()
```

Finally, for the "Add an Item" menu item's script, type this line:

```
Activities.addAccomplishment()
```

This would be a good time to save the root if you're as paranoid as we tend to be about computer systems, power companies, lightning bolts, and the like.

Create Needed Objects

Besides the menu and the scripts, this suite calls for two pieces of data to be stored in its table: an outline called Accomplishments and a number called howMany. You could write your scripts in such a way that if the user runs the suite and these items don't exist yet, they are created automatically. But, as we said at the beginning of the chapter, we aren't out to build commercial software here, so we'll take the easy way out and create the objects in the table so we know they're there when we test our suite.

Create a new number object called howMany. Notice that Frontier obligingly initializes its value to 0, which in this case is exactly what we want. If we wanted it to have some other initial value, of course, that would be easy to assign.

Now create a new outline object called Accomplishments.

You are ready to test your suite.

Testing the Suite

Everyone has their own approach to testing. The main idea is to make sure you've tested all of the functions under as many circumstances as you can think of. Here's one way to test the Tracker suite.

1. In the Quick Script window, type and execute this line to start the suite running: `Activities.installMenu()`
2. From the Tracker menu that displays, select "Edit Menu." Be sure the menubar editor appears, then close it.
3. Select "Edit Table" from the Tracker menu; be sure the table appears, then close it.
4. Select "How Many?" from the Tracker menu. It should display a dialog box telling you that you have added 0 items to the log since we haven't yet added anything.
5. Select "Add an Item" from the menu or type Command-I. When the dialog appears asking you what you accomplished, type something short that you'll remember. Press Return or click the "OK" button.
6. Select "How Many?" from the Tracker menu. Be sure it has updated so that it now shows one item added to the log.
7. Add another item or two, repeating the sequence in steps 5 and 6.
8. Now select "Edit the List" from the Tracker menu. When the list appears, confirm that it has today's date as its first summit (there may be a blank summit at the top; don't worry about it but if it bothers your sense of aesthetics and balance, delete it) and your items under it.

9. Edit the date so that it is yesterday's date (or last Tuesday's or the anniversary date of the last Mets' World Series victory; whatever you like as long as it's not today).
10. Close the outline.
11. Add another item or two to the list.
12. Select "How Many?" from the Tracker menu and satisfy yourself that the counter is indeed keeping accurate track of your life.
13. Select "Edit the List." Now you should see two summits, one with the date you changed the original to at Step 9 and today's date, each with its items beneath it.

That's probably sufficient testing for this suite. Unless you ran into bugs you had to fix, you're nearly done.

Adding the Suite to a Menu

It would of course be terribly inconvenient to type a command into the Quick Script window each time you want to run your Tracker suite. So we'll add a menu item for it to another menu. You can select either the Custom menu or the Suites menu. (Actually, you can add this to any of the customizable menus, but these are the logical repositories for such things and we suggest you stick with them.) We'll put this on the Suites menu.

You already know how to do this, so we'll just zip through it quickly. Open the Suites menu with the Option key held down. Pick the place you want this suite added, position the bar cursor there, press Return, and type "Tracker" (or any other name you like that will remind you of the suite's purpose). Now click on the script button and in the window type this line:

```
menu.addSuite(@Activities)
```

This is the standard way of adding menus belonging to suites to the menubar. Notice that the menu.addSuite verb requires the address of the table whose menu you wish to add. Because the suites sub-table is part of the Frontier path table stored in system.misc.paths, you need not supply a full address to its location in the Object Database.

Testing Launch from Menu

Now you should select the "Tracker" (or whatever you called it) option from the Suites menu and ensure that it displays the right menu and that the menu options all work as expected.

Saving the Modified Root

You can now save the root file in the usual way. You have added a mini-application, or suite, to your Frontier environment. Never again will you have that sense that you've been running in place for the past two years. Just look at all you've accomplished!

Third Example: An Agent Script

For our last example of scripting the Frontier environment, we'll build something smaller than our last example but a little more interesting in terms of its behavior. We'll build an agent script.

A Frontier agent is a script that executes as a background process. Frontier supports a theoretically unlimited number of agent scripts, each associated with its own scheduler and each running in the background. (See Chapter 6 for a discussion of agents and their role in Frontier.)

This simple script will "wake up" every minute and check your people table at a location called nextAppointment. If the time it finds there matches the present time, it will display a dialog reminding you of the appointment and sound the Macintosh's speaker as well.

To run as an agent, a script has to reside in the table system.agents, so open that table and create a new script entry named myReminder. Here's the code for this script:

```
if people.[user.initials].nextAppointment == clock.now ()
    speaker.beep ()
    dialog.notify ("Time for your next appointment!")
clock.sleepFor (60) «Sleep for 60 seconds.
```

The first line compares the date stored at the myReminder spot in the user's table to a string with the present time and determines if they are equal. Notice in the name of the table entry, we use square brackets around user.initials. This causes UserTalk to treat this as an expression and to evaluate it before inserting it into the database table address we are constructing. This is a method for making your routines generic. Rather than including your personal initials in this script and having it therefore be unusable by others without modifications, this approach works with everyone's table, assuming, of course, they have an entry called nextAppointment.

If the two values are equal, Frontier sounds the speaker and puts up a dialog informing you that it's time for your next appointment. If the two times and dates don't match, this agent goes to sleep for 60 seconds and then checks again.

You can probably see how you could use this agent as a starting place to create a reasonably robust alarm system that could keep multiple appointments in a table in the Object Database and remind you when it is time to leave for a meeting or to eat your lunch.

All you need to do to make this agent work for you is an entry in your people table called nextAppointment that is a date type value. When you create it, it will put the current time and date into the value field. You can simply edit this value for the next reminder you want to set.

Agents have all kinds of potential uses. For example, Frontier includes a set of network utilities that automatically backup files on a network at a specified time when nobody is using the system. You could easily write a script that would automatically save your root every hour whether you remembered to do so or not. Let your imagination run; you'll probably come up with a lot of interesting uses for these intriguing little critters.

Sharing Scripts

If you write a UserTalk script of which you are particularly proud, we hope that you will get into a sharing spirit and give copies of it to friends and colleagues. You should also think about uploading a copy of it electronically to one or more of the bulletin board systems

where other Frontier users can see your work and admire it (as well as getting the benefit of the script itself).

Frontier support groups and bulletin boards can be found on CompuServe, America On-Line, and AppleLink. These are not only good places for you to upload your own UserTalk scripts, but also to obtain scripts written by UserLand and by other Frontier users.

HTML reformatting by Steven Noreyko January 1996

Chapter 9

Scripting the Operating System

This chapter discusses one of the most powerful aspects of UserTalk: its ability to manipulate files, folders, and other system-level objects on the Macintosh. Close to 100 of UserTalk's verbs deal with files, folders, the Finder, the system, and launching applications and other tools entirely from within Frontier.

We'll begin this chapter with a brief discussion of the kinds of activities for which UserTalk defines verbs that will assist you in scripting the Finder and desktop activities. Then we'll walk through the construction of two small but powerful examples. The first asks the user to identify a file of which to create an alias. It then creates the alias, places it into the Apple Menu Items folder in the System Folder and, optionally, launches the program for the user immediately. The second example is a desktop script that counts all of the files in a folder or volume that have changed in each month and updates an Object Database table accordingly.

Before reading this chapter, you might want to read or review Chapter 3, which discusses desktop scripts in detail.

The Verb Sets

UserTalk includes a full complement of verbs to deal with the Macintosh Finder and System. These verbs are, of course, fully documented in the UserTalk Reference Guide, but we'll take a look at them here primarily to give you some notion of the scope of things you can ask Frontier to do for you at these levels.

The verbs that deal with system-level activities can be divided into the following categories, which match those in the UserTalk Reference Guide:

- file verbs
- Finder verbs
- launch verbs
- speaker verbs
- system verbs

We'll look briefly at each of these verb types in the following sections.

File Verbs

There are 64 file verbs in UserTalk. They enable you to do such things with files and folders as:

- copy, delete, move, rename, and create them

- find out if they exist, when and by what application they were created, what version they are, when they were last modified, and how big they are
- change their creator and type
- compare two files
- count the lines in a file
- find text in a file's contents
- display standard Macintosh file dialog boxes
- find out how many bytes, files, or folders are in a folder
- read and write text files
- find out how many volumes are on-line, how big they are, how much information is stored on them, how much free space they have, and whether they are ejectable
- break a full file path name into file, folder, and volume information
- lock and unlock files and folders and find out if they are locked
- create alias files, find out if a file is an alias, and follow aliases to their parents
- copy a file to the System Folder

These are just some of the dozens of functions you can perform on Macintosh files, folders, and volumes with the file verbs.

Finder Verbs

The Finder is an application that comes with your Macintosh. (You'd be surprised how many people don't know that!) It deals with the display of your files and folders and with their launching, location, size, and other information.

Finder verbs in UserTalk can be used to handle such tasks as the following:

- changing the way files and folders are viewed
- opening, closing, growing, zooming, and moving windows in the Finder
- duplicating, moving, dragging, opening, printing, putting away, and making aliases of files
- restarting or shutting down the system
- moving Finder to the front of your application
- showing the Clipboard

Launch Verbs

With the launch verbs in UserTalk, you can launch any object that appears in the Apple menu, any Control Panel, any application, or any code resource (such as an FKEY).

Speaker Verbs

You can set up sound parameters and activate the system's speaker with the UserTalk speaker verbs.

System Verbs

System-level verbs deal for the most part with applications that are running. You can use UserTalk verbs to:

- find out if an application is running

- find out how many applications are now running
- retrieve the ID of any running application
- find out which application is frontmost
- move a particular application to the front

Contents Page | Next Section -- **Example One: Alias Maker**

HTML reformatting by Steven Noreyko January 1996

Chapter 9: continued

Example One: Alias Maker

One of the handiest things in Macintosh System 7 is the idea of an alias file. An alias file is a small file that points to the file from which it is spawned. You can put an alias anywhere on the system and it will be able to find its "parent," or original file. Any kind of Macintosh file - application, document, even Control Panel - can have one or more aliases available.

Aliases can be placed anywhere in the system. For example, a document that you might possibly look for in any of three possible places depending on your mind set at the time can now literally be in three places at once. Finding either alias of the file will open it exactly as if you'd found the original.

Many times, you want to store an alias of an application or document you frequently use in the Apple Menu Items folder in the System Folder. This makes that object's name appear on the Apple menu, which means that you can open it from anywhere in the Macintosh system.

Carrying out this process in the Finder is a time-consuming and relatively complex project involving these steps:

1. Finding the file you wish to alias. This usually involves opening one or more folders in the Finder.
2. Choosing "Make Alias" from the File menu
3. Opening the System Folder
4. Locating and selecting the Apple Menu Items folder
5. Dragging the alias file into the Apple Menu Items folder
6. Generally, renaming the file

The UserTalk verb file.newAlias carries out these same tasks with one command. You simply tell the verb what file you wish to create an alias for, and the name of the alias file, including the path name to the Apple Menu Items folder and it takes care of the rest. Here's a typical use of this verb:

```
file.newAlias(\"System:Apps:Hot App\", \"System:System Folder:Apple Menu
Items:Hot App\")
```

Pretty simple, right?

The problem, of course, is that this one-line script isn't very useful if you want to move another new alias file to the Apple menu. You'd have to re-type the full path and file names (or at least substantial parts of them) each time you wanted to do this task. So we'll create a slightly more flexible version of this script.

Our script will ask the user to identify the file of which to create an alias. It will then ask for the name the alias should have in the Apple menu. Finally, it will create and copy the alias using the file.newAlias verb.

First, we need a line that will allow the user to select the file to alias. Looking through the UserTalk Reference Guide, we find that in the File Dialogs section there is a verb called file.getFileDialog. It sounds like the right one. It takes three arguments: a prompt with which to guide the user, the address of the Object Database cell or local variable in which the user's answer is to be stored, and the type of file to be shown.

Here is a line of UserTalk code which will enable the user to pick a file of any type and put its full path name into a variable called originalFile:

```
file.getFileDialog(\"File to create alias for?\", @originalFile, 0)
```

The zero indicates that all file types should be displayed.

Next, we need a line that asks the user for the name of the file to create in the Apple Menu Items folder. We can use dialog.ask for this purpose:

```
dialog.ask(\"Save in Apple Menu under what name?\", file.fileFromPath
(@scratchpad.originalFile))
```

Other than an escape valve for the user to cancel this operation, we are now ready to enter the code. Here is the full listing of the storeNewAlias script:

```
on storeNewAlias ()
    local
        f1 = ""
        f2 = ""
    if not file.getFileDialog ("File to create alias for?", @f1 , 0)
        return (false)
    f2 = file.fileFromPath (f1)
    if not dialog.ask ("Save in Apple Menu under what name?", @f2)
        return (false)
    file.newAlias (f1 , file.getSystemFolderPath () + ":Apple Menu Items:" + f2)
    speaker.beep ()
    dialog.notify ("Done!")
    return (true)
```

In good UserTalk style, we use the "on" keyword to define the script header and then we declare two local variables, initializing them both in this case to empty strings. Next, we ask the user for the file of which to create an alias. Clicking the "Cancel" button in this dialog returns false, resulting in the script being exited and returning false to the calling script.

After we extract only the file name from the full path of the file chosen by the user and put the file name into f2, we use that as the default response to the next dialog, which asks the user for the name to be used in the Apple Menu. Here again, if the user clicks "Cancel," the script terminates and returns false to the calling script.

Finally, we use file.newAlias to create the alias of the file whose full path name is stored in f1 and store it in the right place on the system drive. Notice the use of the UserTalk verb file.getSystemFolderPath, which returns the path to the System Folder on the startup volume. This makes the script generic to all users' systems.

When we're done, we beep the speaker and let the user know we've finished. Then we return true so the calling script will know all went well.

If creating aliases for the Apple Menu is something you do a lot, you might want to connect this new script to a menu. You need not do so, of course.

You might want to try modifying this script so that it will also let the user find a folder to alias and place on the Apple Menu. That is perfectly legal under System 7, but this script won't allow that because we use file.getFileDialog. Be careful, though; just switching to file.getFolderDialog will make it impossible to alias files. You'll probably want to ask at the outset if the user wishes to create an alias of a folder or a file. Check out dialog.threeWay for this one.

Contents Page | Previous Section | Next Section -- **Second Example: Logging Changed Files**

HTML reformatting by Steven Noreyko January 1996

Chapter 9: continued

Second Example: Logging Changed Files

The example we'll discuss in this section is a desktop script. It is designed to measure periodic activity on a volume or in a folder by recording in a Frontier Object Database table the number of files that changed in each of the 12 months of the year.

To begin with, we need to build a table to record the information. We'll take advantage of the existence of a table in Frontier that almost meets our needs. Jump to system.verbs using the "Jump To..." option on the UserLand menu or its Command-J keyboard equivalent.

Select the sub-table called constants but don't open it. Copy it from the table with Command-C.

Now open your people table and paste the newly copied sub-table in the usual Macintosh manner. Select the name of the table, "constants," by double-clicking on the Name entry in the people table, and change it to "months."

Having created a log table for the information we're about to gather to be stored, we're ready to write the script. This script will accomplish the following tasks:

- Zero out all of the entries in the log table to be sure we don't accumulate data.
- Loop through all of the files in the folder and sub-folders in which the desktop script is stored, examining their last-modified date, extracting the numeric value of the month, and updating the appropriate entry in the log table.
- Inform the user of completion and, if requested, open the log table for the user to review, print, or edit.

Here is what the script looks like fully expanded so we can examine every line:

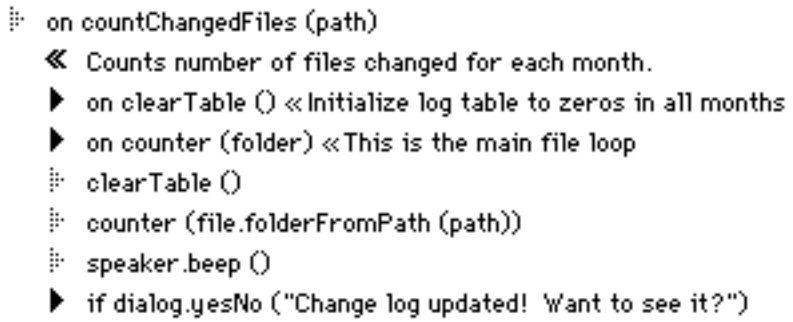
```
on countChangedFiles (path)
    « Counts number of files changed for each month.
        « Stores result in a table using array indexing.
    on clearTable () «Initialize log table to zeros in all months
        for i = 1 to 12
            people.[user.initials].months [i] = 0
    on counter (folder) «This is the main file loop
        fileloop (f in folder)
            rollBeachBall ()
            if file.isFolder ()
                counter (f)
            else
                modDate = file.modified (f)
                date.get (modDate, @day, @mo, @yr, @hr, @min, @sec)
                people.[user.initials].months [mo] ++
    clearTable ()
    counter (file.folderFromPath (path))
    speaker.beep ()
    if dialog.yesNo ("Change log updated!  Want to see it?")
        edit (@people.[user.initials].months)
        Frontier.bringToFront ()
```

After initializing a local variable, we write two local scripts. The first is clearTable. It uses a "for" loop to make sure all of the log table's entries are zero. (Note that we're using square brackets to tell Frontier we want to use the value stored in the table user.initials as the second item in the table address.) The second local script is counter. It should look familiar; we copied it from the desktop script countWordFiles, which we saw in Chapter 3, and then modified it slightly for our new purposes.

The interesting part of the counter script is the "else" clause. It uses the UserTalk verb file.modified to obtain the date the current file was last modified. Then it applies the date.get verb to this result to convert its value to a set of numbers reflecting the day, month, year, hour, minute, and second of the modification date. We are really only interested in the month, but date.get converts all of the elements at once. Finally, it uses array indexing on the log table, indexing it by the month value we just got from date.get and incrementing the value of that location.

The main part of the program consists of the last few lines. The first two of these lines call the local scripts we just discussed. The last three lines notify the user that the process is complete and ask if he wants to see the log table. If so, we display it for editing and bring Frontier to the front, making it the active application. Otherwise, we're finished.

Incidentally, here's how the script looks when we collapse groups of lines together. We recommend you store your scripts this way to permit the less knowledgeable user to be able to see at a glance the general flow of your script without having to delve into its inner workings.



You can test this script from within Frontier by typing the command countChangedFiles and a valid path name for your system into the Quick Script window. Here's an example:

```
countChangedFiles("Hard Disk:System:System Folder:")
```

Once you've satisfied yourself that this script works, you need to export it to its proper place: the desktop. You can do this by typing Command-3 or by selecting "Export a Desktop Script..." from the Export sub-menu of the Edit menu.

Frontier will ask you to confirm that you wish to export the currently selected object to the desktop. Then it will give you a chance to put the desktop script where you like. Remember from our discussion about desktop scripts in Chapter 3 that these scripts work within the scope and context of the volume or folder on or in which they are stored.

After you've found a place for your desktop script, you should test it.

To run a desktop script, the user simply double-clicks on it in the Finder. If Frontier is not running, it will be launched.

HTML reformatting by Steven Noreyko January 1996

Chapter10

Scripting for IAC

One of Frontier's major strengths is its ability to work with other applications. It can interact with external applications in two basic ways:

- It can drive a single application, acting as that application's scripting language.
- It can act as an intermediary to allow several IAC-aware applications to talk to each other by giving them a common ground through which to interact.

In this chapter, we will focus on how to write UserTalk scripts that take advantage of IAC-aware applications. Our perspective will be that of the UserTalk script writer who is scripting IAC-aware applications. If you are interested in creating IAC-aware programs or in making a current application IAC-aware, UserLand Software produces the Frontier Software Developers Kit (SDK) to facilitate the process of creating verbs (or, as we sometimes call them, "wires") in an application. (Frontier SDK is included with Frontier and has full documentation.) Once such connections exist, Frontier can be used by UserTalk script writers to control the application.

We will begin this chapter by describing at a high level how your UserTalk scripts can "drive" an application using inter-application communication techniques. Then we'll build a couple of examples of such scripts. These sample scripts enable you to control an IAC-aware application called BarChart that is included in the UserLand Utilities Folder of your Frontier disks.

How Scripts Control Programs

Before you can write a UserTalk script to control an external application (i.e., one other than Frontier), that application must meet two basic criteria:

- It must be IAC-aware (which means that it is programmed so that it is able to receive messages from other applications).

- It must have special scripts in Frontier that enable UserTalk to communicate with its IAC connections. These scripts are called "install" scripts. A special Frontier Object Database table holds one such script per verb for each application that has been installed so that your version of Frontier.root can communicate with it.

  Once an application has fulfilled these two requirements, you can write UserTalk scripts to cause it to do anything for which it has a defined verb. Figure 10-1 shows what actually happens when your UserTalk script instructs an external, IAC-aware application to do something which it "understands" (meaning that it has a corresponding verb or IAC wire and an entry in the Frontier table where its interface is defined).

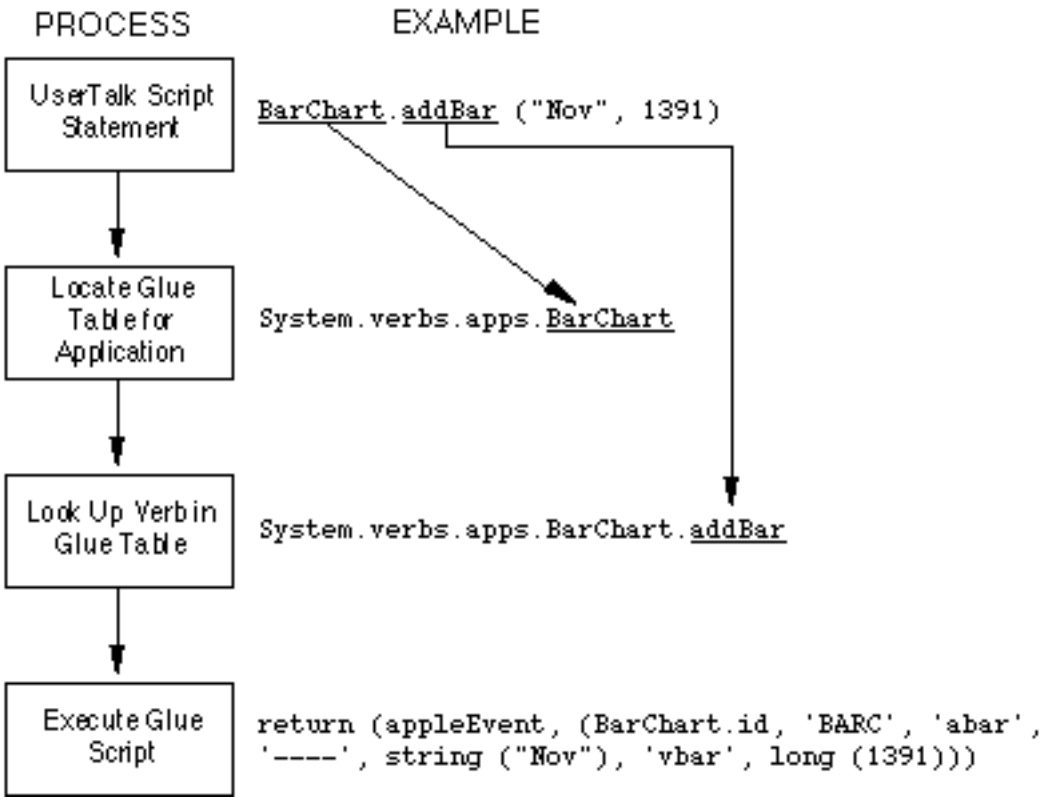


Figure 10-1. Controlling an Application from Frontier

Frontier Install Scripts

Applications that support IAC via Frontier will generally come with a Frontier install script usually named something like BarChart.Frontier (the application name appearing before the period following "Frontier"). When you install the application for the first time, you will simply double-click on this install script. It will then install a table of verbs into Frontier's Object Database. Some install scripts may be available from sources other than the publisher of the application, including on-line bulletin board systems and Macintosh user groups.

As a UserTalk script writer, you will generally not have to be concerned with the inner operations of install scripts for external applications you wish to control. But just to satisfy your curiosity, let's take a quick look at where they are stored and what they look like.

Figure 10-2 shows part of the table system.verbs.apps where the install scripts are stored for applications with which your copy of Frontier can communicate.

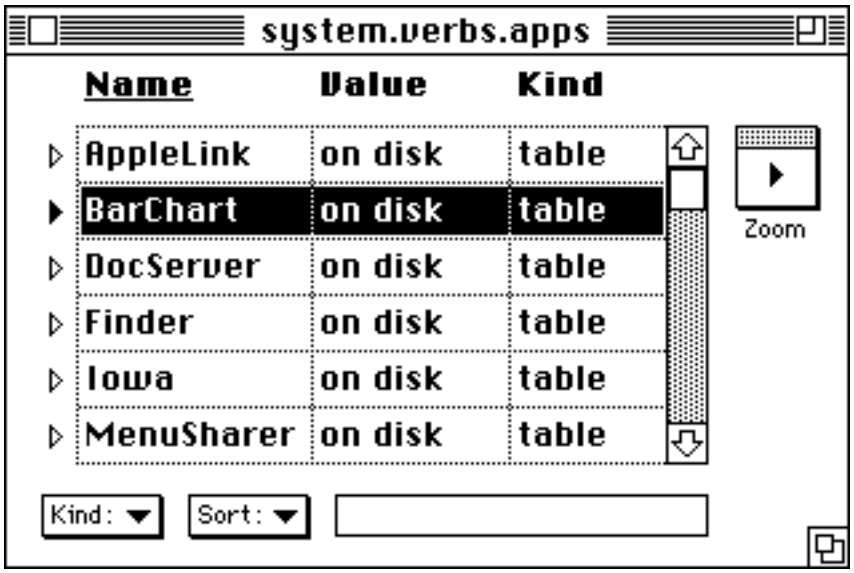


Figure 10-2. The Install Scripts Table in Frontier

Let's open the install table for BarChart. It looks like Figure 10-3.

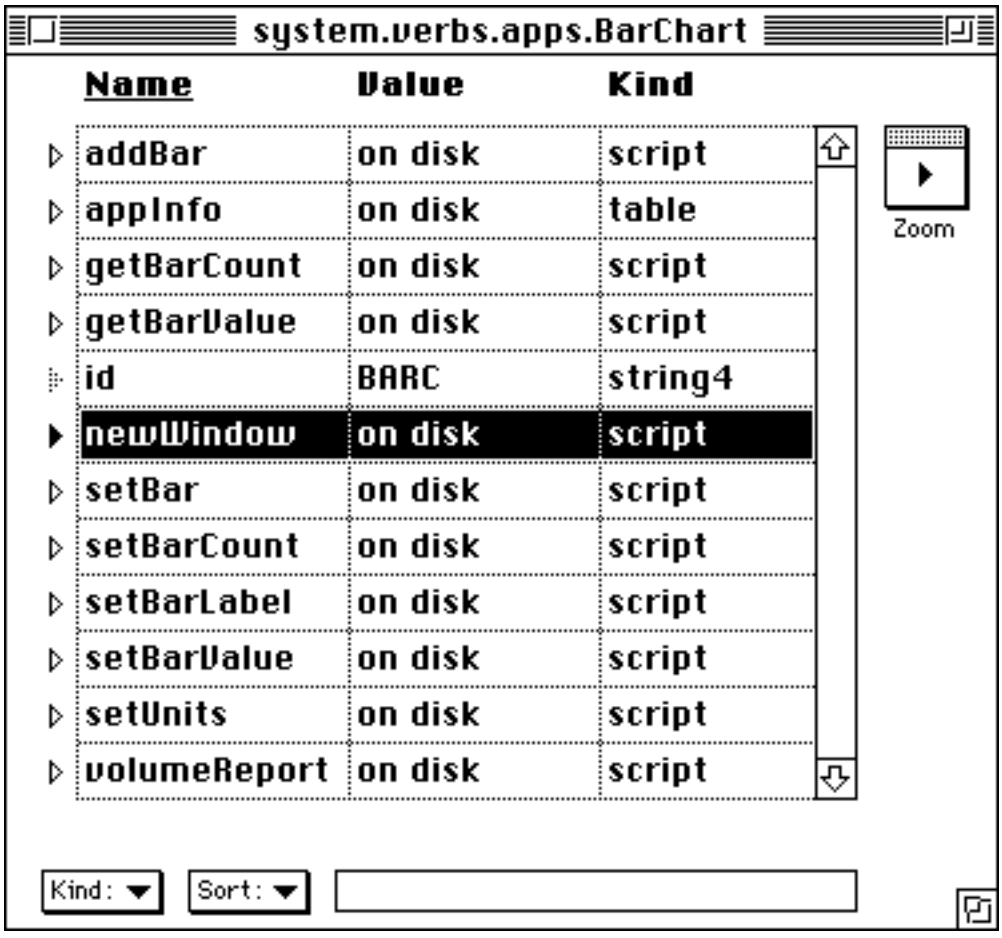


Figure 10-3. BarChart Install Table

You can probably determine what most of these install scripts will let you tell BarChart to do. Documentation for these verbs is in the Frontier SDK folder, inside the BarChart folder.

Just for fun, let's take a look at one of these install scripts. Double-click the item marker next to the script newWindow. The script looks like this:

```
on newWindow (ctbars, title, units) «create a new BarChart window
    if not app.newWindow (title)
        return (false)
    if not BarChart.setBarCount (ctbars)
        return (false)
    return (BarChart.setUnits (units))
```

This is a fairly normal-looking UserTalk script, not unlike many others we've seen in this book. It uses a generic newWindow verb in the app table to create a new window, then makes sure it is able to set up a window to accommodate the desired number of bars and units. Notice that this script calls BarChart.setBarCount. Let's open that install script by Command-double-clicking on the verb's name in the newWindow script. It looks like this:

```
on setBarCount (ctbars) «reset the number of bars frontmost BC window
    return (appleEvent (BarChart.id, 'BARC', 'scnt', '----', short (ctbars)))
```

Now this script looks a good bit different from other scripts we've seen, at least in its operational line. The first line is familiar-looking. The second line is typical of install scripts in Frontier. It uses a call to Frontier's appleEvent verb (which handles Apple Events), passes it some information about what the verb wants to do and what application it wants to use, and returns the result of that call.

You can close those two script windows now. You'll never have to look at another install script unless your curiosity simply overwhelms you.

Contents Page | Next Section -- **Using Install Scripts**

HTML reformatting by Steven Noreyko January 1996

Chapter10: continued

Using Install Scripts

Now that we know what install scripts look like, how do we use them in our own UserTalk scripts? The short answer is that there is no fundamental difference between a UserTalk script that drives an external application and one that manipulates the Operating System or adds functionality to Frontier itself.

To use a verb connected to an IAC-aware application, you write a UserTalk script that:

- ensures that the application is both running and selected as the current application with which to conduct IAC dialogs
- calls one or more of that application's verbs using the same notation we've been using throughout this manual

To use a BarChart verb, then, you simply call it by providing the application's name, followed by a period, followed by the verb name, followed by a set of parentheses enclosing its arguments, if any. We saw examples of such calls when we looked at sample install scripts.

Let's launch BarChart and take a look at one of its menus to see how we can implement its verbs in a real-life situation. Select "Launch BarChart" from the "Apps" sub-menu of the "Custom" menu in Frontier. The application will launch. Because BarChart supports a special Frontier protocol called "menu sharing," we can examine the scripts attached to its menus. In BarChart, hold down the Option key while you select "Folders on Each Disk" from the "Utilities" menu in BarChart. Hold the Option key down until Frontier opens the menubar editor for the menu, then click on the "Script" button. The resulting script looks like this:

```
BarChart.volumeReport ("Folders on Each Disk", " folders", @file.foldersOnVolume)
```

As you can see, this menu option simply calls the volumeReport verb in BarChart and provides three arguments: the name of the window to open, the label string to be used on the bars in the graph, and the name of the verb to execute for each volume. If you're curious how this works, you can Command double-click on the last argument to open the file.foldersOnVolume verb or you can simply run the script by returning to BarChart and choosing the menu option. If you do run the script, the window will probably look something like Figure 10-4.

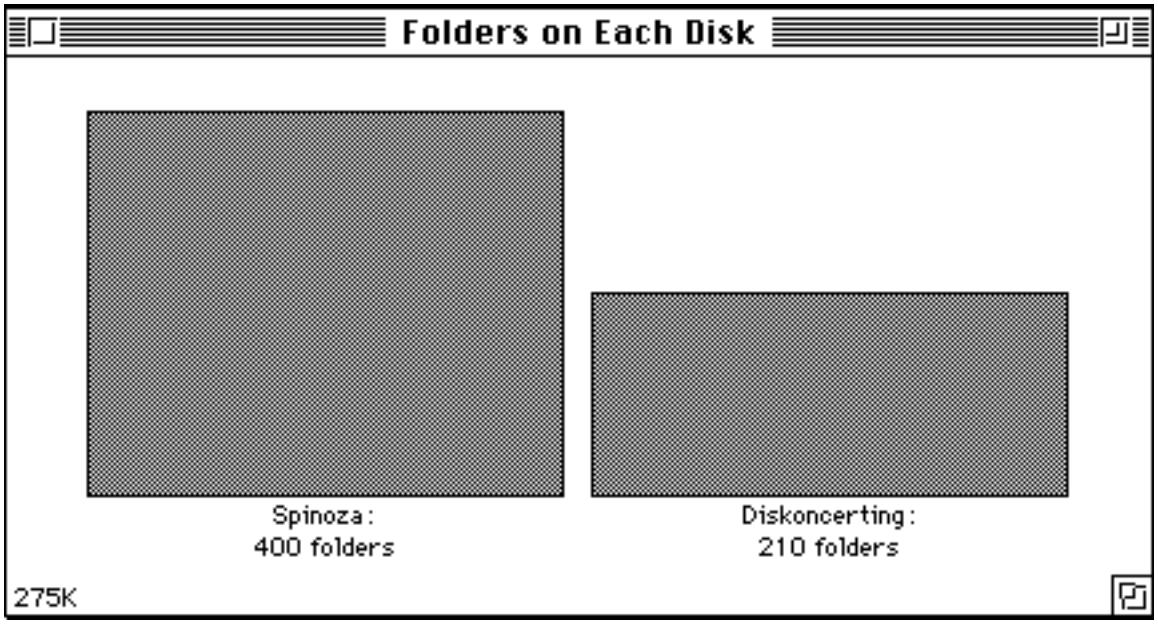


Figure 10-4. Sample BarChart Output from "Folders on Each Disk" Option

As you can see, using external applications from within Frontier is for all practical purposes as easy as using Frontier itself. As final proof, let's build a couple of useful examples that use BarChart and Doc Server.

Just because we're using a UserLand-supplied program in these examples, don't get the impression that only applications built by UserLand or applications that use our special Frontier SDK or other "magic" can be used this way. We don't have any way of knowing for sure which of the growing number of IAC-aware applications you have on your disk, but we do know you have BarChart, because we gave it to you! The fundamental principles we'll be following in these examples apply to any commercial or other software you want to control from UserTalk.

Example One: Charting File Changes

In Chapter 9, we built a desktop script that loops through a disk or folder and records the number of files that changed in each month of the year. That script is called countChangedFiles. Recall that we saved it as a desktop script, so it can be run from the desktop by the user's simple acts of placing the icon in the folder or disk he wants to log and double-clicking it.

In this example, we're going to demonstrate three things:

1. You can execute a desktop script from inside its own Frontier window exactly as if it were a non-desktop script.
2. Driving an IAC-aware application from Frontier is quite straightforward.
3. You can re-use scripts in UserTalk quite readily.

Making a Desktop Script Accessible for Debugging

The desktop script countChangedFiles can be run from inside Frontier exactly as it is from the desktop. Just type the script and a path name into the Quick Script window, like this:

```
deskscripts.countChangedFiles (\"Hard Disk:System:\")
```

If we want to debug this script, we can't do so. Why? Because the script is designed to be called externally, but we want to debug it in its own window. To prove we have a problem, open the script countChangedFiles and click on the "Debug" button. You will see a dialog box informing you that this can't be done. This is because the script consists entirely of one local script that starts with the keyword "on." There are no script statements that actually tell the script to do anything. UserTalk scripts that are meant to be called externally are usually formatted this way. Desktop scripts must be written this way. So we'll make this script into one that we can run in its own window, and thus debug. Doing so is quite simple. Go to the end of the script, create a new heading, and use Shift-Tab to move to the summit level of the outline. Now type this line.

```
countChangedFiles (\"Hard Disk:System:\")
```

Compile your change. Now click on the "Debug" button and notice that the debugging options appear.

This demonstrates our first point: desktop scripts can be easily changed to be executed from within Frontier rather than the desktop.

HTML reformatting by Steven Noreyko January 1996

Chapter10: continued

Driving an IAC-Aware Application from UserTalk

Now let's move to our second point and demonstrate how easy it is to drive an IAC-aware application from UserTalk.

To write a script that drives an application whose IAC wires have been attached to Frontier with a glue table, we only have to know two things:

- what we want to accomplish
- what verb(s) we need to carry out our wishes in the target application

We want to translate the numeric data stored in the log table we've created with the countChangedFiles script into a nice readable chart in BarChart. There should be one vertical bar per month and each bar should reflect the total number of files modified in each month. In essence, then, we probably need to:

1. Make sure BarChart is running and is the IAC destination.
2. Open a window in BarChart into which to put the new chart.
3. Loop through the log table and draw a bar for each entry in the table.
4. Bring BarChart to the front so we can see our handiwork.

If you look at the table system.verbs.apps.BarChart, you can probably spot the verbs you need to use. If we use newWindow to open a window large enough to accommodate 12 bars, then we can use setBar to draw each bar. (Note that you might at first have thought that using addBar would be appropriate, but a careful read of its description indicates that it adds a bar rather than drawing a data value in an existing bar.)

Here, then, is a script that will accomplish the tasks we wish to carry out:

```
on chartChangedFiles ()
    if not (app.start ("BarChart")) «Make sure BarChart is IAC target
        return (false)
    local (logTable = @people.[user.initials].months)
    BarChart.newWindow (12, "File Activity Chart", 1)
    for i = 1 to 12 «Chart data in 12-row people.XXX.months table
        BarChart.setBar (i, string.mid (nameOf (logTable^[i]),1 ,3), logTable^[i])
    sys.bringAppToFront ("BarChart")
chartChangedFiles ()
```

We think you'll agree that this is a straightforward script, yet it is doing something quite powerful. It is transforming data in an Object Database table into a nice chart produced by another application that was created independently of Frontier and then wired to it. Figure

10-5 shows the result of a typical run of this script.

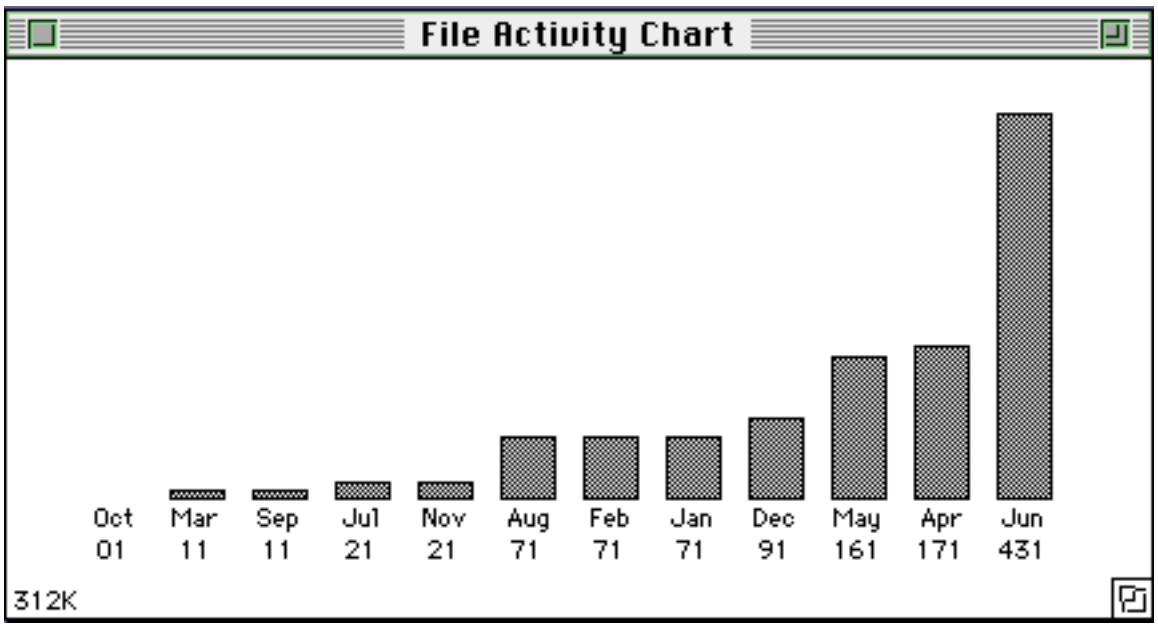


Figure 10-5. Typical BarChart Output

As you can see, this is a much more readable way of representing the data in the table, which in turn is extracted from a survey of the desktop.

Reusing UserTalk

Code reuse is one of the most important concepts in programming. When you have to program a particular task more than once, you are wasting the time to write the code as well as the space each copy occupies on the hard disk. You are also creating a potential maintenance headache. If you discover a better way of doing some particular task, you may be a lot less inclined to add it to the system if the code it affects is scattered throughout the system.

By combining the countChangedFiles script with the one we just built, we can create a new script. This script will ask the user if the log table is up to date. If not, it will get the user's input on what to log and then do so. In either case, it will then automatically produce a chart from the contents of the log table. Here's the script that does these activities:

```
on chartChangedFiles ()
    if not dialog.yesNo ("Is the log table up to date?") «Need to run other script
    if not app.start ("BarChart") «Make sure BarChart is IAC target
    local (logTable = @people.[user.initials].months)
    BarChart.newWindow (12, "File Activity Chart", 1)
    for i = 1 to 12 «Chart data in 12-row people.XXX.months table
    sys.bringAppToFront ("BarChart")
chartChangedFiles ()
```

Let's open the first outline level in this script and examine it so you can see how we've reused the script from the last chapter. Here's what that portion of the script looks like:

We use two dialogs to determine from the user if the log table is current and, if not, whether the user wants to run a new log table on a whole disk or on a folder. Then we use Frontier's verbs that display the standard Macintosh file dialogs to have the user point to the disk or folder to log. Finally, we simply call the script we constructed in Chapter 9, passing the location of the path.

Here is the full script of the chartChangedFiles mini-application:

```
if not dialog.yesNo ("Is the log table up to date?") «Need to run other script
    ans = dialog.twoWay ("Run log on disk or folder?", "Folder", "Disk")
    case ans
        1
            file.getFolderDialog ("Which folder to log?", @scratchpad.x)
        2
            file.getDiskDialog ("Which disk to log?", @scratchpad.x)
    system.deskscripts.countChangedFiles (scratchpad.x)
```

Example Two: Full-Text Searching

Besides BarChart, Frontier comes with another IAC-wired application called DocServer. We talked briefly about this application in Chapter 7. As you might expect, since UserLand wrote DocServer, it is wired for IAC. In this section, we'll see how to create an application that adds functionality to an IAC-wired application beyond what it was originally designed to be able to do.

As DocServer is shipped, you can look for a specific verb with the Command-J option or the "Find a Verb..." menu equivalent. But what if you want to find a string that isn't a verb name? DocServer doesn't incorporate this capability, but thanks to the verbs it does include, we can build this new functionality in via Frontier scripts. Further, we can make this new operation available directly in the application thanks to menu sharing.

First, we'll modify the menu so that the new full-text search capability is available from the DocServer menu in the application.

Then we'll write a script that enables us to carry out full-text search. We'll attach this script directly to the menu option with which it is associated.

```
on chartChangedFiles ()
    if not dialog.yesNo ("Is the log table up to date?") «Need to run other script
        ans = dialog.twoWay ("Run log on disk or folder?", "Folder", "Disk")
        case ans
            1
                file.getFolderDialog ("Which folder to log?", @scratchpad.x)
            2
                file.getDiskDialog ("Which disk to log?", @scratchpad.x)
        system.deskscripts.countChangedFiles (scratchpad.x)
    if not app.start ("BarChart") «Make sure BarChart is IAC target
        return (false)
    local (logTable = @people.[user.initials].months)
    BarChart.newWindow (12, "File Activity Chart", 1)
    for i = 1 to 12 «Chart data in 12-row people.XXX.months table
        BarChart.setBar (i, string.mid (nameOf (logTable^[i]),1 ,3), logTable^[i])
    sys.bringAppToFront ("BarChart")
chartChangedFiles ()
```

Modifying the Menu

DocServer supports the UserLand Frontier menu-sharing protocol, so at least part of its menubar is defined in Frontier and encoded into UserTalk scripts. You've undoubtedly had plenty of experience modifying menus, so we'll walk through this part of our example quickly.

Select the "Jump" command from the UserLand menu (or type Command-J, its keyboard shortcut) and jump to menubars.DOCS.

Position the bar cursor on the "Jump to a Verb..." line and press Return. Now type a new menu item name, "Find a String...". (You can immediately confirm that DocServer picked up this change, even if DocServer was running at the time, by opening the DocServer application and examining its menu.)

Open the script of the new menu item by clicking on the "Script" button in the menubar editor.

HTML reformatting by Steven Noreyko January 1996

Chapter10: continued

Starting the Process

First, let's make sure DocServer is the IAC destination application. Then let's ask the user to give us the string to find and store the answer in the DocServer table. By storing the value in the database, we make it possible for the search string to be remembered from one search to the next. To make typing easier, we'll put the answer into a local variable as well. Then we'll initialize two more variables that we'll need later. Here's the first part of the script that handles these tasks:

```
app.start ("DocServer")
if not app.askDialog ("What do you want to find?", @scratchpad.x)
    return (false)
s = scratchpad.x
local (count = DocServer.getVerbCount (), a = True)
```

Notice that we have used a DocServer verb - getVerbCount - to find out how many verbs are in the database associated with DocServer.

Looking Through the Table

Now we just want to set up a loop that goes through all of the verbs in the database and looks for the string for which the user is searching. But first, we have to figure out where in DocServer the verbs exist that will enable us to read the text of a verb and then try to figure out what form the text is in when it comes into Frontier.

Open the system.verbs.apps.DocServer table and browse through it. You will notice a verb called getVerbText. Open its script. You may be able to tell by reading it that this verb makes a call to something called a complex event (which is beyond the scope of this manual). Its first argument is another entry in the same table, called verbText. Since the verb is designed to get some information, it is probably logical to assume that this argument might be the address in which it stores the information it locates. To confirm that, Command-double click on DocServer.verbText in the script window. The table that appears should look something like Figure 10-6.

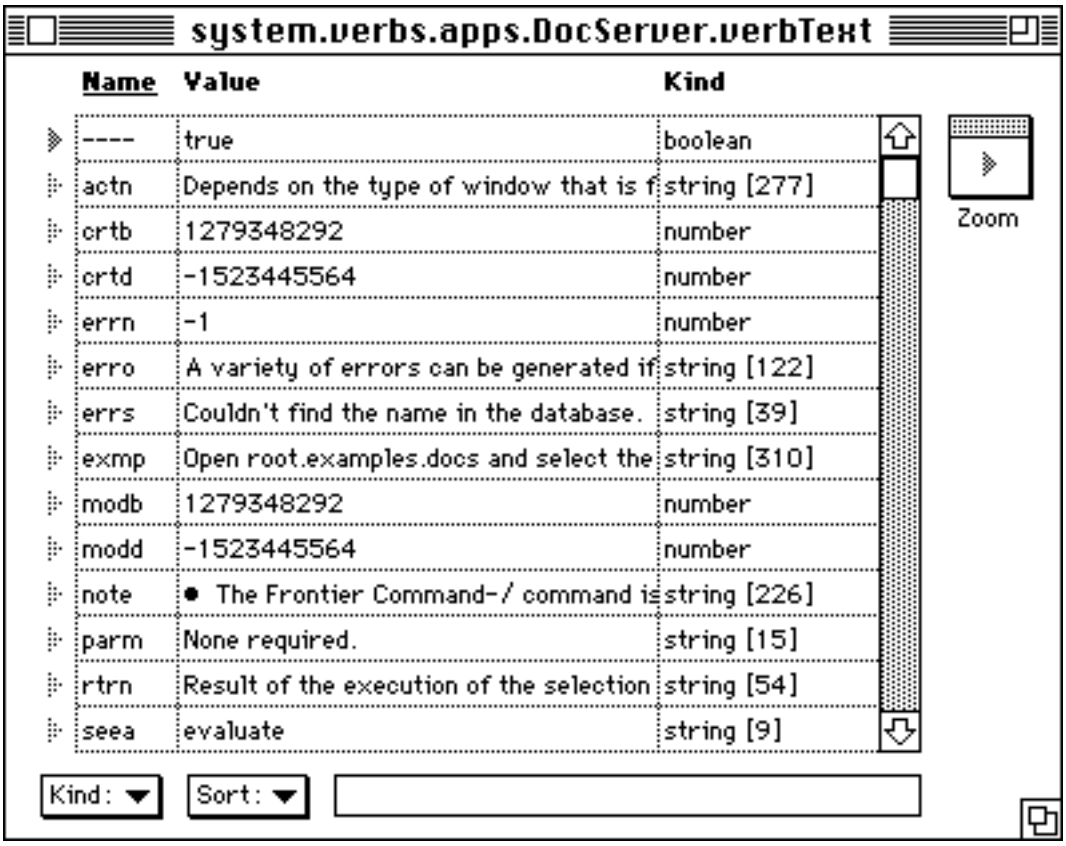


Figure 10-6. Partial Contents of DocServer.verbText

As you can probably deduce from reading the "Value" fields, some of these objects contain textual information from the description of the verb as it appears in the UserTalk Reference Guide. You could browse through this table and find out for yourself which table entries relate to text content in the verb description and which don't, but we'll do that for you. These are the entries that interest us:

- actn (Action section of the verb description)
- erro (Errors section of the verb description)
- exmp (Examples section of the verb description)
- note (Notes section of the verb description)
- parm (Parameters section of the verb description)
- rtrn (Returns section of the verb description)
- seea (See Also section of the verb description)
- synt (Syntax section of the verb description)

These entries of interest don't arrange themselves in any predictable way within the table; they're sorted alphabetically by name. But they all contain strings and, as it happens, there is only one string in the table that isn't of interest. So we'll just go through the table looking for entries that are strings (i.e., those that have a Frontier datatype of "stringType") and then see if the search text is in each of those strings.

Here is the code for that loop:

```
if count > 0
    verbName = DocServer.getFirstVerb ()
    for i = 1 to count «Loop through all the verbs
        DocServer.getVerbText (verbName)
        « Check each entry in the verbText table for a match
        for j = 1 to sizeOf (DocServer.verbText)
            if typeOf (DocServer.verbText[j]) ≠ stringType
                continue
            if string.patternMatch (s, DocServer.verbText [j])
                showFind ()
                if not a « User doesn't want us to keep going.
                    return (true)
        verbName = DocServer.getNextVerb (verbName)
```

Dealing With a Match

We first check to be sure that we have found at least one verb in the DocServer database. If so, we go to the first verb (to give us a known starting point) and put its name into a variable called verbName. Now we set up two nested loops. The first loops through all of the verbs using a counter variable called "i." The second loops through each entry in the verbText table looking for the search string. It uses a counter variable called "j." Within the second loop, we look at each entry in the verbText table. If it's a string entry, then we look at its contents using the Frontier verb string.patternMatch. If we locate the string, we call a local script called showFind, which we'll examine next. If the entry isn't a string, we use continue to move to the next table entry. If we find a match, we call showFind and then check the contents of a variable called "a" (which we haven't defined yet but which you can probably see will be part of the showFind script's task).

Let's Take a Look at ShowFind

This straightforward local script uses DocServer's built-in displayVerb to show the verb in which the search text was located. It then puts up a dialog box using Frontier's app.ConfirmDialog verb asking if the user wishes to continue the search. The user's reply is stored in the variable "a." Recall from the previous script fragment that we use this value to determine whether to stop processing. The app.ConfirmDialog verb returns true if the user clicks "OK" and false if he clicks on "Cancel."

Here is the entire script for this verb so you can check your version against ours.

```
on showFind () «Show the find and ask if we should continue
    DocServer.displayVerb (verbName)
    a = app.confirmDialog ("I found this one. Find the next one?")
app.start ("DocServer")
if not app.askDialog ("What do you want to find?", @scratchpad.x)
    return (false)
s = scratchpad.x
local (count = DocServer.getVerbCount (), a = True)
if count > 0
    verbName = DocServer.getFirstVerb ()
    for i = 1 to count «Loop through all the verbs
        DocServer.getVerbText (verbName)
        « Check each entry in the verbText table for a match
        for j = 1 to sizeOf (DocServer.verbText)
            if typeOf (DocServer.verbText[j]) ≠ stringType
                continue
            if string.patternMatch (s, DocServer.verbText [j])
                showFind ()
                if not a « User doesn't want us to keep going.
                    return (true)
        verbName = DocServer.getNextVerb (verbName)
```

Contents Page | Previous Section | Next Chapter -- **Appendix**

HTML reformatting by Steven Noreyko January 1996

Appendix

Glossary

[a | b | c | d | e | g | h | i | m | o | p | q | r | s | t]

1-based
Indexes can be 0-based or 1-based. If they are 1-based, as all of UserTalks indexes are, the first element in a list or array is element number 1. In zero-based indexes, the first element is element number 0.

a

address
A shorthand way of describing an objects location in the Frontier Object Database. It consists of one or more terms. If more than one term is needed to identify an objects address, the elements of the address are separated by periods. Each segment of an objects address represents a sub-table with the possible exception of the last segment, which is of a datatype appropriate to the element being addressed.

agent
A UserTalk script that executes as a background task. All such scripts are stored in the table system.agents. Agents execute repeatedly, triggered as often as called for by their script contents or every second in the absence of contrary instructions in the script.

Apple Event
Messages sent from one application or process to another within the Apple Macintosh system. Apple Computer has defined and cataloged a great many such events, but individual application developers also define their own. UserTalk has special code to deal with Apple Events generically.

b

background script
See agent **bar cursor**
The highlight that indicates the selection of an entry in a Frontier outline-type object (outline, script or menubar). The bar cursor highlights the entire visible portion of the entry on which it is positioned (i.e., the entire line).

Boolean

A datatype that can have a value of true or false. Name derives from a historical figure in computing.

built-in
A verb is said to be built-in if it is included as part of UserTalk as it is delivered by UserLand Software.

bundle
A programming construct in UserTalk that permits you to group a number of related lines under a single heading so that they can be collapsed. The keyword bundle does not itself have any execution significance. It provides a means of organizing scripts.

bundle-ize
To create a bundle of lines in a UserTalk script.

c

callback
A script that is executed repeatedly as a result of the use of one of UserTalks verbs that iterates over a group of windows or table entries. See the verbs op.visit and table.visit in the UserTalk Reference Guide for examples.

case
A programming construct that facilitates choosing among two or more alternative courses of action depending on the value of a variable or the outcome of a calculation.

character
A single ASCII character enclosed in single quotation marks.

chevron
A character made by holding down the Option key and pressing the backslash key and used to delineate comments in UserTalk scripts. It looks like this: .

d

de-bundle
To reverse a bundle operation.

de-hoist
To reverse the effects of a hoist operation in a Frontier outline-type object (outline, script, or menubar) **desktop script**
A UserTalk script designed to be executed from the Finder desktop by double-clicking on it. Such scripts generally operate on files and/or folders. Their scope is usually confined to the folder or volume in which they are stored, and all sub-folders and their contents. A desktop script is created by exporting a UserTalk script using the Export a Desktop Script... option

on the UserLand menu.

e

event
A message sent by one program to another or by the user (via actions such as mouse clicks and menu selections) to a program. Events are at the core of the Apple Event based model for interapplication communication.

f

factoring
The process of defining the component scripts and event processors into which to divide a particular group of related or potentially related functions. Proper factoring of a UserTalk script into several small scripts can increase reusability of the components.

fileloop
A UserTalk keyword that initiates a loop that iterates over a collection of files and/or folders.

Frontier.root
The default name for the file containing the Object Database. More than one root file may be defined and opened at one time. Root files need not be named root so long as they were created as root files. The root is a database file whose contents must be explicitly saved as opposed to such a file whose contents are continuously updated as changes occur as is the case with traditional database files.

h

heading
An entry in an outline type object (outline, script, or menubar) consisting of one line indicated by an item marker. Headings may be either summits or sub-headings.

headline
See heading **hoist**
A process which is applicable to an outline type object (outline, script, or menubar) heading in which the immediate sub-headings of the selected heading appear to be converted to summits. The display is changed to show only those sub-headings and their sub-headings. This has the practical effect of limiting editing to a selected portion of the outline. This operation can be undone with de-hoist operations.

i

IAC
An acronym that stands for interapplication communication, which in turn

is the process of two or more applications transferring information and commands between or among one another. There are numerous protocols for IAC on various operating systems. On the Macintosh, IAC is supported primarily via the manipulation of Apple Events. Frontier is designed so that it can support all IAC approaches.

item marker
The right-pointing triangle that appears at the start of every line of every Frontier outline-type object (outline, script, or menubar). A solidly black item marker means a heading has one or more collapsed and invisible subheads. A gray item marker means the heading has no collapsed subheads (in other words, any sub-headings it does have are visible).

m

Main Window
The Frontier window which is associated with a Frontier.root file. It has two display modes. In one, only a message area, close box, popup menu, and flag are visible. In the other, the flag is activated to reveal four buttons in a second area immediately beneath and attached to the Main Window. Closing the Main Window has the effect of closing a root file.

o

Object Database
All of the information stored in or required by Frontier and UserTalk is stored in the built-in Object Database. This database is displayed to the user as a series of tables and sub-tables, each of which can in turn contain either another table or an object of any Frontier datatype. From a scripting standpoint, the Object Database can be thought of as an extensible global variable storage area.

outline window
A Frontier window in which an outline, script, or menubar is displayed.

p

parameter
Sometimes referred to as an argument. A value that accompanies a verb to provide additional information about the execution of the verb. Verbs that require parameters generally use them as either focusing information (aiming the verbs action at a particular object or file, or example) or as an object on which to perform some operation.

path
A Macintosh file path is a string consisting of the name of the volume on which the file or folder is stored, followed by a colon, followed by the name of the folder in which the object (folder or file) being sought is stored or the name of the folder or file itself. This path name description of an object can continue arbitrarily deeply, but the Macintosh imposes a 255-character

limit on the entire name of the path to any object in the Operating System.

protocol
A set of agreed-upon methods by which a process involving two or more entities takes place. Specifically in Frontier and IAC, the suite of verbs and events which together make up the agreed-upon process by which two or more applications communicate with one another. The Macintosh uses Apple Events as the main concept in its IAC. Other operating systems use different approaches (for example, Microsoft Corporation is implementing an IAC protocol called Object Linking and Embedding, or OLE, on its Windows platforms).

q

Quick Script Window
The Frontier window in which the user can type UserTalk scripts and execute them immediately by pressing the Enter key or the Run button in the window itself.

r

root
The main table in the Frontier Object Database. This table is always called root regardless of the name of the database file of which it is part. See Frontier.root.

s

scratchpad
Generically, an area of memory or disk space designed for transient use in a computer system. Frontier is equipped with a sub-table of root called scratchpad for which you may find frequent use in UserTalk scripting.

string
A Frontier datatype consisting of one or more characters enclosed in double quotation marks.

string4
A Frontier datatype consisting of exactly four characters enclosed in either single or double quotation marks. This type of data is used to designate a Macintosh files creator and type as well as resource types.

subhead
A heading in an outline object (outline, script, or menubar) which is subordinate to another is said to be a subhead of its parent heading.

sub-heading
See subhead.

suite

A collection of one or more UserTalk scripts (usually at least two scripts are involved but this is not a requirement) with related functionality. Suites can be thought of as miniature Frontier applications which can be loaded and unloaded by the user at will. Suites are typically stored in the Object Database table called suites, a sub-table of the root table.

summit
Also referred to as summit heading or summit-level heading. An outline heading at the highest (left-most) level of an outline. A Frontier outline object (outline, script, or menubar) can have more than one summit. See subhead.

t

table
Generically, a two-dimensional array consisting of one or more rows, each of which contains one or more columns. In Frontier, all objects in the Object Database are stored in tables consisting of one or more rows of three columns each. The three columns are Name, Value, and Kind.

table window
A Frontier window in which a table object is displayed and in which it may be edited.

target
The Frontier object to which targeted verbs apply. Verbs which rely on the existence of a target and operate solely on the target are delineated in the UserTalk Reference Guide. The target is, by default, the frontmost window in Frontier but it can be explicitly changed with UserTalk verbs.

target window
The window containing the object which is the current target for UserTalk verbs.

text window
A Frontier window in which a word processing text object is displayed and in which it may be edited.

Contents Page

HTML reformatting by Steven Noreyko January 1996