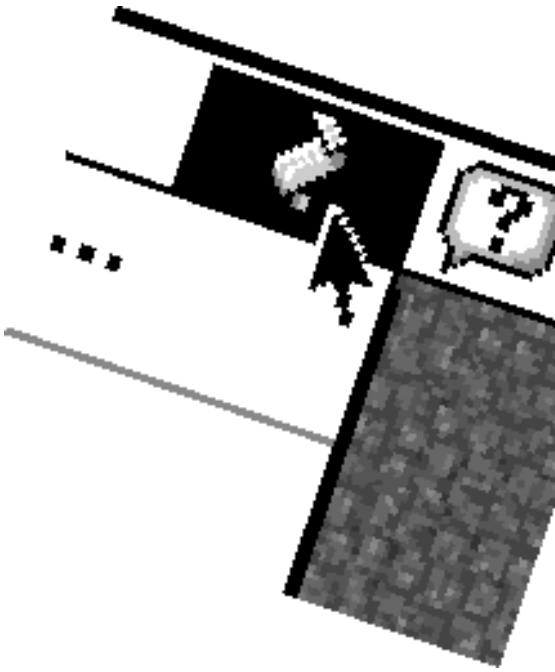
Apple Confidential

The Scripting Menu

Unsupported 1.0d9 Experimental Version
11 November 1992



Application Scripting Group
Developer Tools
© 1992 Apple Computer, Inc.

Apple Confidential

Contents

Contents.....ii

Introduction..... 3

Product Information.....4

Availability and Audience.....4

System Requirements.....4

Overview.....5

Installation.....5

What’s In the Menu.....5

Keyboard Equivalents.....6

Theory of Operation.....7

What It Patches.....7

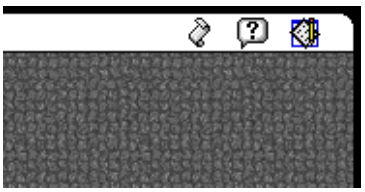
How a Folder Belongs to an Application.....7

How It Runs Scripts.....7

The Run-Script Event.....8

Apple Confidential

Introduction



The Scripting Menu

The **Scripting Menu** is an iconic system menu (like the Help menu) that appears on the right side of the menu bar whenever a scriptable[1] application is active. Its menu items are scripts: choose an item from the menu and the script will run. You can even attach keyboard equivalents to these scripts and execute them from the keyboard alone. Some commands are always available, and some are only available within particular applications.

Generally the target of the script is the active application, and scripts that operate on the user selection are particularly useful. Scripts in the Scripting Menu act like extensions to the command set. of the application.

For example: Word has a nice feature that applies consecutive numbers to a range of paragraphs. Quill doesn't. But by writing a script to do just that, and adding it to Quill's scripting menu, you can give Quill this feature.

The main advantage to scripts in the menu is that they're very lightweight. You can call them up with one mouse click (or keystroke) without leaving your current context for the Finder.

> **Important:** The Scripting Menu is still just an investigation. It's not part of any product and has no official go-ahead. We're still investigating the thorny issues of user-definable menus and menu commands, and this is probably not exactly how it will work or what it will look like!

[1] Actually, right now it appears in any Apple event aware application. This will be fixed.

Availability and Audience

The scripting menu is not an official part of the AppleScript product. If it does ship, it'll do so in a different form. It'll be part of an AppleScript release targeted at end users.

The Scripting Menu will appeal to all users. It's easy to use and understand, and looks like MacroMaker. Having scripts available inside applications, and having those scripts operate on the selection or the active window, allows scripts to act just like extra application menu commands. People can use simple utility scripts to increase their productivity.

"Solution" scripts that tie together databases, forms packages, spreadsheets, *etc.* and perform complex processing with them are great. But the low end or casual user —who can't run five applications at once, or doesn't need to — will still find the Scripting Menu usable and useful. And even the massive "Run My Entire Business" script can be triggered from a menu item…

System Requirements

Of course, scripting components will need to be installed. There is also an extension (INIT) file, which patches two traps and installs an Apple event system handler. The extension presently uses about 20k of memory in the System heap.

Running a script requires having enough free memory to instantiate its scripting component. The component allocates 8k from the application heap when it starts, and may allocate more as needed while the script is running. When the script finishes, the component is closed and all its memory is

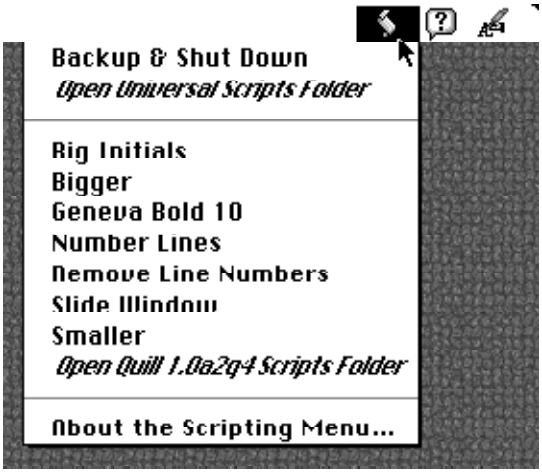
freed.

OVERVIEW

Installation

The Scripting Menu Extension needs to be placed in the Extensions folder. That's it. The outer "Scripts" folder will be created when the extension first installs itself; other script folders are created on demand.

What's In the Menu



The Scripting Menu, pulled down

The menu has three parts:

- On top are the universal scripts, which always appear in any scriptable application.
- Below those are the scripts for the active application.
- At the bottom are the fixed commands. Right now there's only one: the "About the Scripting Menu" command that displays an About box.

At the end of each set of scripts is an italicized command that, when chosen, tells the Finder to open the folder for those scripts. For instance, when Quill is active there is an "Open Quill Scripts Folder" command that opens the "Quill Scripts" folder.

These folders are like the Apple Menu Items folder: any script files placed in them immediately[2] appear in the scripting menu. (They must be compiled scripts, not text files or applets.)

◆ *Right now these application script folders, and the "Universal Scripts" folder, are stored in a central "Scripts" folder in*

[2]Right now the menu doesn't update until you switch applications. This will be fixed.

the System folder. One of our main topics of debate is whether it would be better for each application's scripts folder to appear in the same folder as the application itself. Many extendable applications, like Canvas and Word, already have these kinds of sub-folders.

Keyboard Equivalents

It will be a great boon to be able to assign hot keys to the commands in the Scripting Menu. Everyone loves hot keys. The user should be able to assign any keystroke, not just the usual command keys, and iconic representations of the keys involved should appear in the menu. MacroMaker did a great job of this.

You can assign an ordinary command key to a menu command by appending to the name of the script file a square root symbol (“√”, option–v) followed by the command character. This appendage doesn’t appear in the name of the menu command, but the character becomes the command key equivalent. This is not a very good interface, but it is suffient to experiment with this capability.

THEORY OF OPERATION

What It Patches

- It head-patches DrawMenuBar. Whenever the active application has changed since the last DrawMenuBar, it rebuilds the menu for the new active app (or deletes it, if the app is not scriptable.)
- It head-patches SystemMenu to intercept menu commands in the scripting menu.
- It installs an Apple event system handler for its run-script event.

As of version d4, about 16k of locked INIT code is loaded into the System heap, and about 2k of relocatable icons and other data.

How a Folder Belongs to an Application

Each script folder contains a four-letter signature of its matching application. (The Universal Scripts folder has the signature '****'.) The signature is stored in an invisible file whose name is “Signature” followed by a CR character. For efficiency’s sake, it’s not stored in the file’s data or resource fork. Instead, OR-ing together the file’s type and creator codes results in the signature. (The signature is scrambled in this way to prevent the signature file from being found in a disk-search for files of a given type or creator.)

How It Runs Scripts

The SystemMenu patch gets the menu command. It finds the script file and loads the 'scpt' resource from it.

At this point it could just fire up an OSA component and run the script. However, the application is inside a call to MenuSelect() or MenuKey() and some applications don’t appreciate having their event handlers triggered while in such a context.

What the patch does instead is to bundle the script data into an Apple event and send it to the active application (but *not* in send-to-self mode. The event goes to the head of the event queue.) Then it returns.

The application then receives the event and calls AEProcessAppleEvent(). This triggers the system handler that was installed by the extension, which extracts the script data, opens an OSA component, and runs the script. Since the application is processing an Apple event, it's expecting its handlers to be called, so all is well.

The Run-Script Event

The event has class 'smnu' and ID 'runs'. The direct parameter is of type 'scpt' and contains generic OSA script data. There's also an optional 'file' parameter, which should be coercible to an FSSpec. This parameter contains the location of the script's file, which is useful if an error occurs and the user wants to edit the script.

- *There's no reason someone else couldn't generate the same kind of event and send it to an application. The system handler provides a general service that will run any OSA script sent to it. Could this be dangerous? If so, I can make the handler refuse to run the script unless the event was sent by the same application that's handling it.*