Nerd's Guide to Frontier

Dave Winer
UserLand Software

Frontier

This article is a top-level technical overview of the UserLand Frontier scripting system, written for experienced C or Pascal programmers. The goal is to show how the language and the environment work, but not to be a complete tutorial in using Frontier.

Frontier is an integrated collection of development tools built around a scripting language and disk-based storage system. Frontier provides a script editor/debugger, table editor, menubar editor, documentation tools and a comprehensive set of built-in verbs that allow you to customize and automate the Macintosh file system, operating system, networks, utilities and scriptable applications.

Frontier scripts can be saved to the Finder desktop, can be linked to menu items, and can run in the background. Scripts can also be embedded in a small application to allow collections of files, folders and disks to be dropped onto the script.

Frontier 1.0 shipped in January 1992; version 2.0 shipped in October 1992. All the examples in this article work with Frontier 2.0.

In the following sections we break out each of the major features in Frontier and discuss them using sample scripts to demonstrate the features.

The UserTalk Scripting Language

UserTalk is a full-featured language, with looping, if-then-else, case statements, local and persistent variables, subroutines, error recovery and automatic type coercion. UserTalk's syntax is most like C or Pascal. The language is very tightly integrated with Frontier's object database, discussed in the "Storage System" section, below.

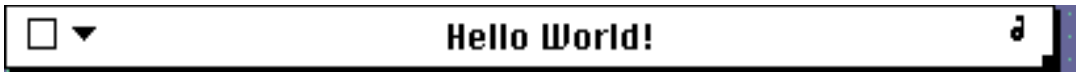
The goal of the language is to make it easy to write utilities that operate at the system level; launching and communicating with applications, managing the file system and operating system and other system resources, and moving information around a network.

Hello World

There's a long tradition of introducing languages with a simple "Hello World" program. Here's what Hello World looks like in UserTalk:

```
msg ("Hello World!")
```

This displays the string in Frontier's main window:



Built-in Verbs

Let's look at a more comprehensive script that creates aliases of all applications on all hard disks in the Apple Menu Items folder:

```
local (appleFolder = file.getSpecialFolderPath ("", "Apple Menu Items", true))
fileloop (f in "", infinity) «scan over all disks, to infinite depth
    if file.type (f) == 'APPL' «it's an application
        local (name = file.fileFromPath (f)) «copy the file's name into a local
        if dialog.yesNo ("Create alias of "" + name + "" in Apple Menu Items folder?")
            file.newAlias (f, appleFolder + name + " alias")
```

The script loops over all files, and when it finds one whose type is 'APPL', it displays a "yesNo" dialog asking if you want to create the alias. If you click on Yes, the script calls the Frontier built-in file.newAlias verb to create the alias in the Apple Menu Items folder.

"fileloop" is a special construct in UserTalk, it allows you to iterate over all the files on a disk or in a folder. If the path is the empty string, fileloop will loop over all mounted disks. By saying we want to go to infinite depth, the loop will visit all files in all sub-folders, no matter how deeply nested.

We could have hard-coded a path to the Apple Menu Items folder, but by calling file.getSpecialFolderPath this script will work on any Macintosh, in any country. For example, in the Fredonia version of System 7, this call will return "Sturgeon:Smidgadzchen Festerest:Apfel Menu Gethingies". (With apologies to the good citizens of Freedonia…)

file.getSpecialFolderPath, file.type, file.fileFromPath, dialog.yesNo and file.newAlias are examples of calls to built-in verbs. Generally, if the Macintosh OS provides an API for a system-oriented operation, Frontier provides a simple scripting API for that operation.

As examples, Frontier 2.0 includes built-in verbs that allow you to:

- Launch an application, data file, control panel, or desk accessory. Loop over all running applications. Bring an application to the front. Access the desktop database.
- Move, copy, delete or rename files and folders. Get and modify file/folder attributes such as the creation date, modification date, file type and creator, size, file comment, version information, icon position. Determine whether a file is visible or not visible, locked or unlocked, busy or not. Reconcile the changes between two versions of the same folder.
- Manage the resource fork of any file. Read and write text files and data files. Access the system clock. Move data in and out of the clipboard.

Frontier is itself completely scriptable and includes built-in verbs to manage its object database script, outline, text and picture windows.

Interapplication Messaging

Scripts that drive applications look very much like scripts that drive the file system and operating system.

Here's a script that creates a StuffIt archive containing compressed versions of all files on all disks modified after May 15, 1993:

```
local (archive = "System:Desktop Folder:Changed Files.sit")
if file.exists (archive)
    file.delete (archive)
StuffIt.launch ()
StuffIt.newArchive (archive, false) «create a 3.0-format archive, not 1.5.1
StuffIt.bringToFront ()
fileloop (f in "", infinity)
    if file.modified (f) > date ("May 15, 1993")
        StuffIt.stuffItem (f)
StuffIt.closeArchive (archive)
```

The new archive appears on the desktop. We delete the file if it already exists. Then we launch StuffIt Lite 3.0, create the new archive, and loop over all files on all disks. If a file's modification date is greater than May 15, 1993, we add the file to the archive. When the loop completes, we close the archive.

The verbs StuffIt.newArchive, StuffIt.stuffItem, and StuffIt.closeArchive result in an Apple Event being sent to the StuffIt application. But this fact is invisible to the script writer. You call StuffIt from a script exactly as if you were calling a built-in verb. The only difference is that you have to launch the StuffIt application in order to call it.

Here's what the script for StuffIt.stuffItem looks like:

```
on stuffItem (path) «return true if StuffIt was able to add the file to the current archive
    return (appleEvent (StuffIt.id, 'SIT!', 'Stuf', 'path', string (path)) == 0)
```

More information on the built-in appleEvent verb is included in the Q&A section at the end of the article.

Object Model Scripting

Some applications implement the richer “object model” style of Apple Events. Scripts that drive object model applications can look very different from scripts that drive simpler scripting APIs, as illustrated in the previous example.

Here’s a script that opens a FileMaker Pro 2.0 database containing information about the 50 states in the United States. It creates a text file that lists each state and its capital on a separate line:

```
local (textfile = "System:States Text", database = "System:States Database")
file.new (textfile) «create the file, its length is 0 bytes
file.setType (textfile, 'TEXT') «it's a text file
file.setCreator (textfile, 'ttxt') «it can be opened by TeachText
file.open (textfile) «open the data fork of the file
app.startWithDocument ("FileMaker", database)
with FileMaker, objectModel «use FileMaker and object model vocabularies
    local (i)
    show (record [all]) «select all records
    for i = 1 to count (layout [1], record) «loop over all the records
        local (stateName, stateCapital)
        stateName = get (record [i].cell ["Name"])
        stateCapital = get (record [i].cell ["Capital"])
        file.write (textfile, stateCapital + tab + stateName + cr)
file.close (textfile)
FileMaker.quit ()
```

In this example, we create a text file on the disk named System, set its type and creator, and open its data fork. We launch the FileMaker application, telling it to open the States Database.

We access FileMaker’s terminology by including the FileMaker-related script code inside the “with” statement. When we refer to count (layout [1], record) we’re asking FileMaker for the number of records when viewed thru the first layout. We make sure all the records are selected. Then we loop over all the records in the database, copying the state name and capital into locals, then writing a line to the text file. After looping over all the records, we close the text file and quit the FileMaker application.

Script editor

Frontier has a full-featured, integrated script editor.

Here's what the first sample script looks like in a Frontier script editing window:

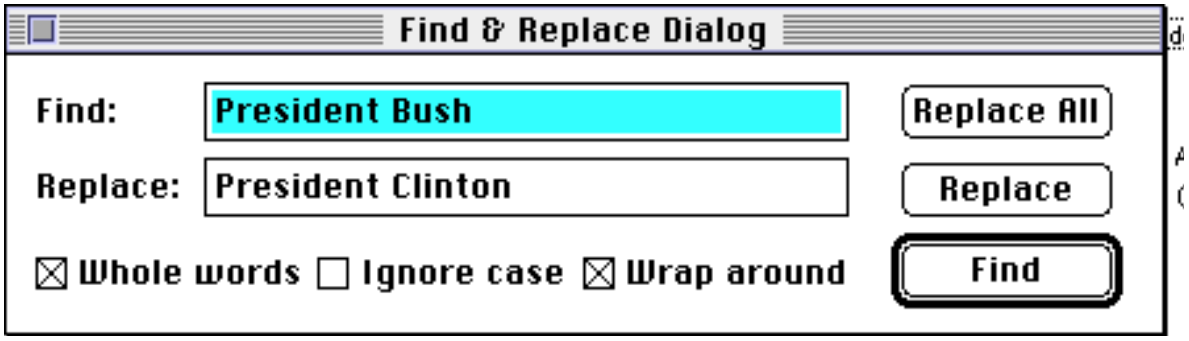
Frontier scripts are outlines. You could collapse the fileloop statement to show none of its sub-heads, or one of them, or all of them. You can control the level of detail you want to view. Outlining also helps you edit the structure of your scripts. Drag a headline to a new location, and all the sub-heads move with it. Programs are hierarchies. Outliners are hierarchy editors. It makes a lot of sense to have a script editing tool that understands hierarchic program structure.

One of the fallouts of this design is that structure symbols such as curly braces and semi-colons are unnecessary when you edit a script using Frontier's script editor. The structure of the outline is the structure of the program and vice versa.

The window title shows the object database address for this script. It's located in the deskscripts sub-table of the system table. Its name is appsToAppleMenu. Details on the object database are in the "Storage System" section, below.

This screen shot was taken using a development version of Frontier that supports multiple scripting components. To the left of the horizontal scrollbar is a popup menu that allows you to select the scripting component to run the script. This allows you to edit an AppleScript script within Frontier and call it from any other script. In fact, you can edit a script in any OSA-compatible scripting language. This includes a new version of CE Software's QuicKeys macro utility, currently in development.

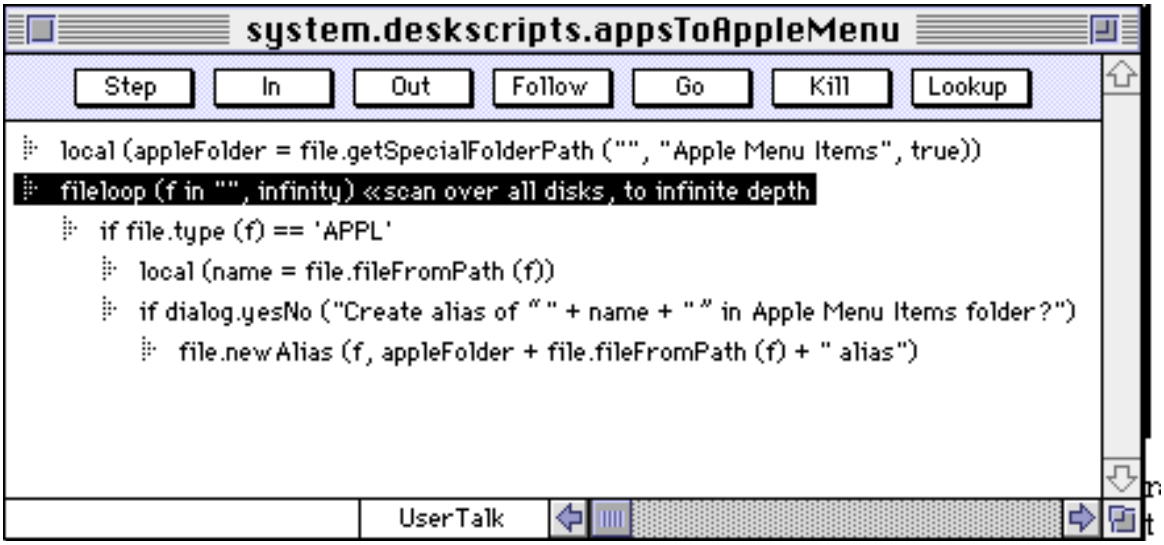
You can find and replace within a single script and over groups of scripts. Options include case insensitive searching, wrap around, find only language identifiers or whole words. Here's a screen shot of Frontier's Find & Replace window:



The script editor is itself scriptable, so you can write tools that automate script development.

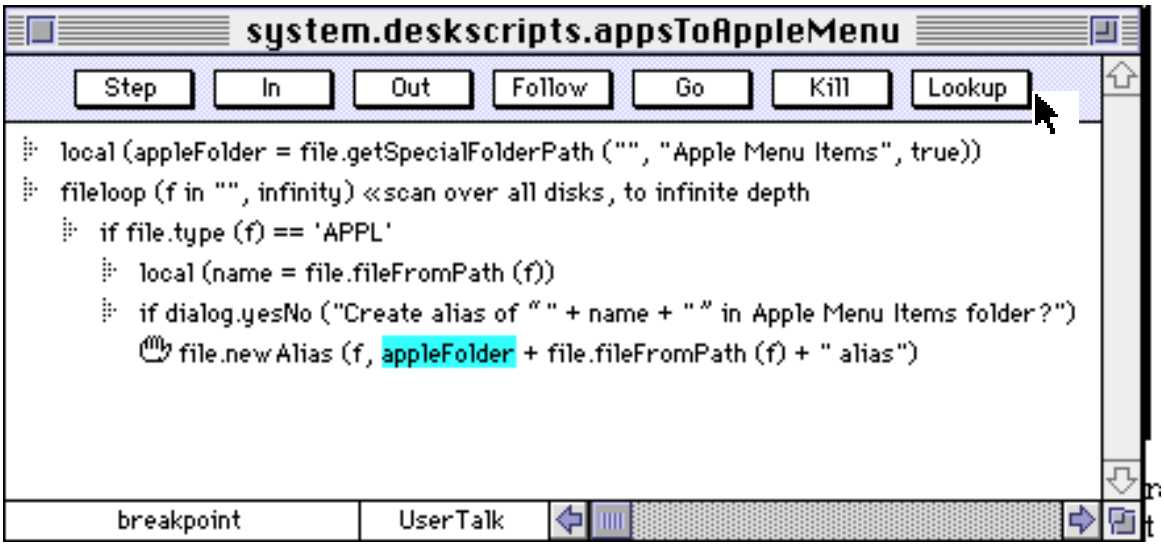
Debugger

The script editor also provides the interface for debugging. Here's what a script editing window looks like after you click on the Debug button:



The buttons at the top of the window are handled by Frontier's integrated script debugger. You can step from statement to statement, go into a script call, step out from a script call, follow statement execution, resume normal execution, or halt the script. You can examine and edit all local and global variables in the storage system while any number of scripts are running. You can set a breakpoint at any statement.

For example, if you set a breakpoint on the file.newAlias call, and clicked on Lookup with the name appleFolder selected, as shown below:



You'd see the top-level stack frame for this script. This table is fully editable:

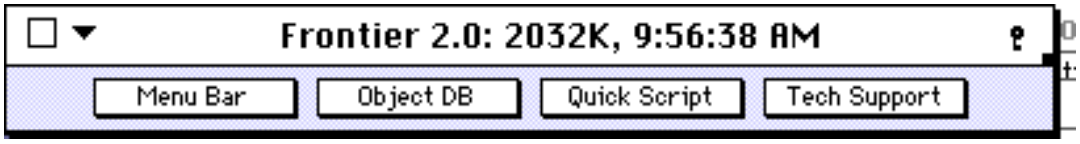
To resume running the script, bring the script window to the front and click on Follow or Go.

When a syntax or runtime error occurs, you can jump to the line where the error occurred, with all script editing features enabled. You can view all local and global variables before terminating the script.

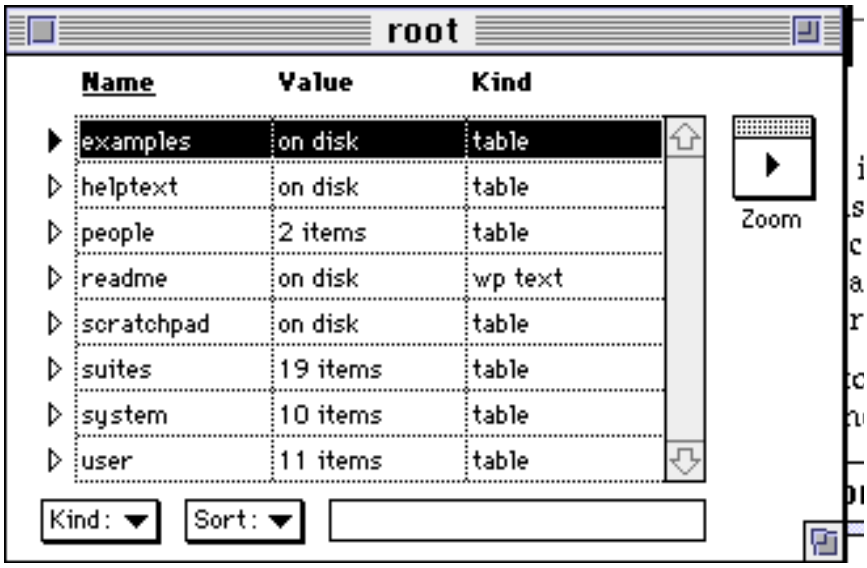
Storage System

Frontier's built-in storage system is called the object database. It can store small objects like a user's name, or the time of the last backup; or larger objects like a list of users, a standard form letter, or a table of electronic mail accounts. It makes it easy for scripts to communicate with each other, especially scripts running in different threads. The object database is a permanent data structure, so you don't have to worry about complicated file formats for your scripts.

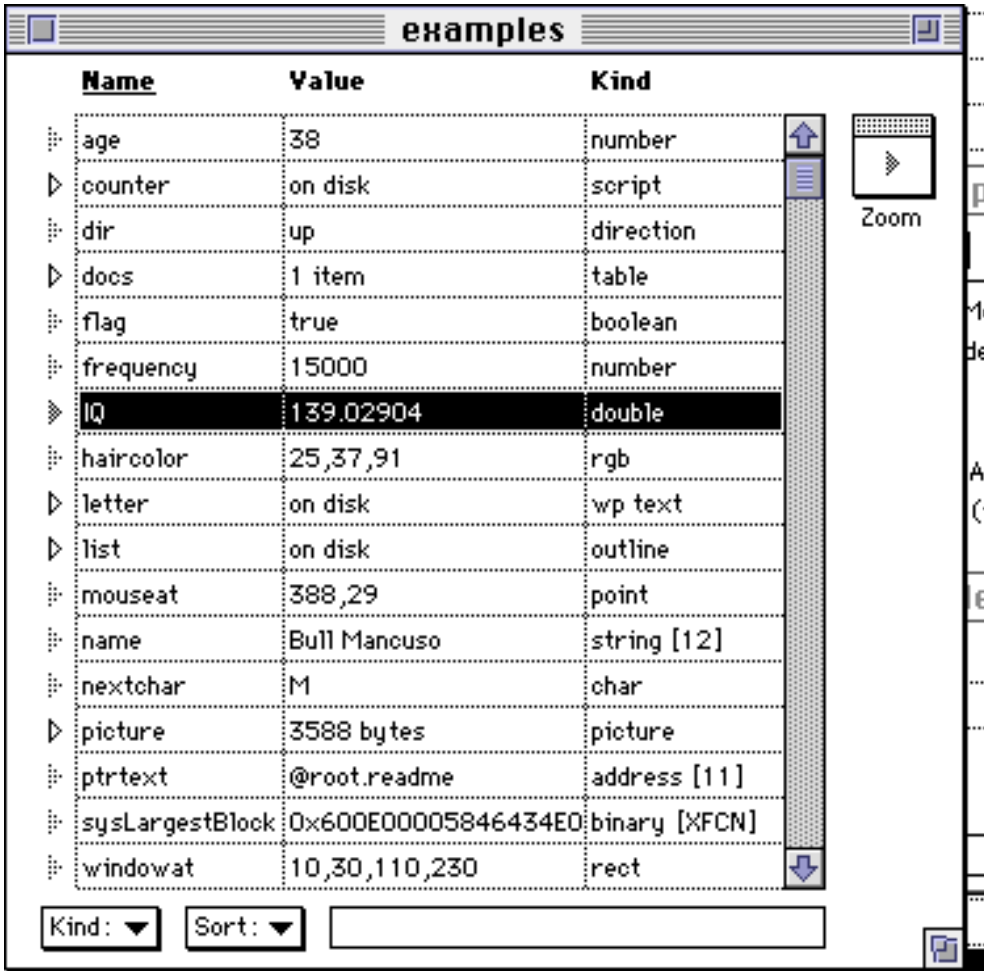
The object database starts with a top-level table is called "root." Start by clicking on the flag in the Frontier's main window. It reveals four buttons:



Click on Object DB button. Frontier opens the "root" table:



There are eight items at the top level of the database. Some are still on disk. Frontier only reads in tables as they are needed. If you double-click on the wedge next to examples, the sub-table opens:



Values can be small things like booleans, characters and numbers; or large things like strings, word processing text, outlines or scripts. Frontier supports over 20 built-in types, and has a general type called “binary” which allows you to store types which Frontier doesn’t directly support.

The Kind popup menu allows you to change the type of an object. The Sort popup allows you to change the order in which objects are displayed in the window.

The storage system uses a 31-bit internal address, so files can be huge, limited only by available disk space. Even though the database is disk-based, Frontier has a Save command, and even a Revert command, allowing you to roll back to the previous version of the database.

From scripts, you use dot-syntax to traverse the table hierarchy. For example, to add 1 to the number “age” stored in the “examples” table you say examples.age = examples.age + 1. You could also say examples.age++.

Because database paths can get long, UserTalk has a “with” statement, like Pascal’s, that lets you easily work with deeply-nested tables. The following script launches MacWrite, brings it to the front, opens and prints a document and quits:

```
with system.verbs.apps.MacWrite
    launch ()
    bringToFront ()
    printDocument ("System:Business Plan")
    quit ()
```

Frontier has another mechanism to help simplify deeply nested values, the paths table. Like Unix or MS-DOS, the paths table defines a set of object database tables that are globally accessible. Because system.verbs.apps is in the paths table, it’s possible to re-write the previous script as:

```
MacWrite.launch ()
MacWrite.bringToFront ()
MacWrite.printDocument ("System:Business Plan")
MacWrite.quit ()
```

Another special table is system.agents. Any script in system.agents is executed once per second. Here’s a very simple agent that adds 1 to a counter once a second and displays the result in Frontier’s main window:

```
msg (scratchpad.count++)
```

If you want to count once a minute:

```
msg (scratchpad.count++)
clock.sleepFor (60)
```

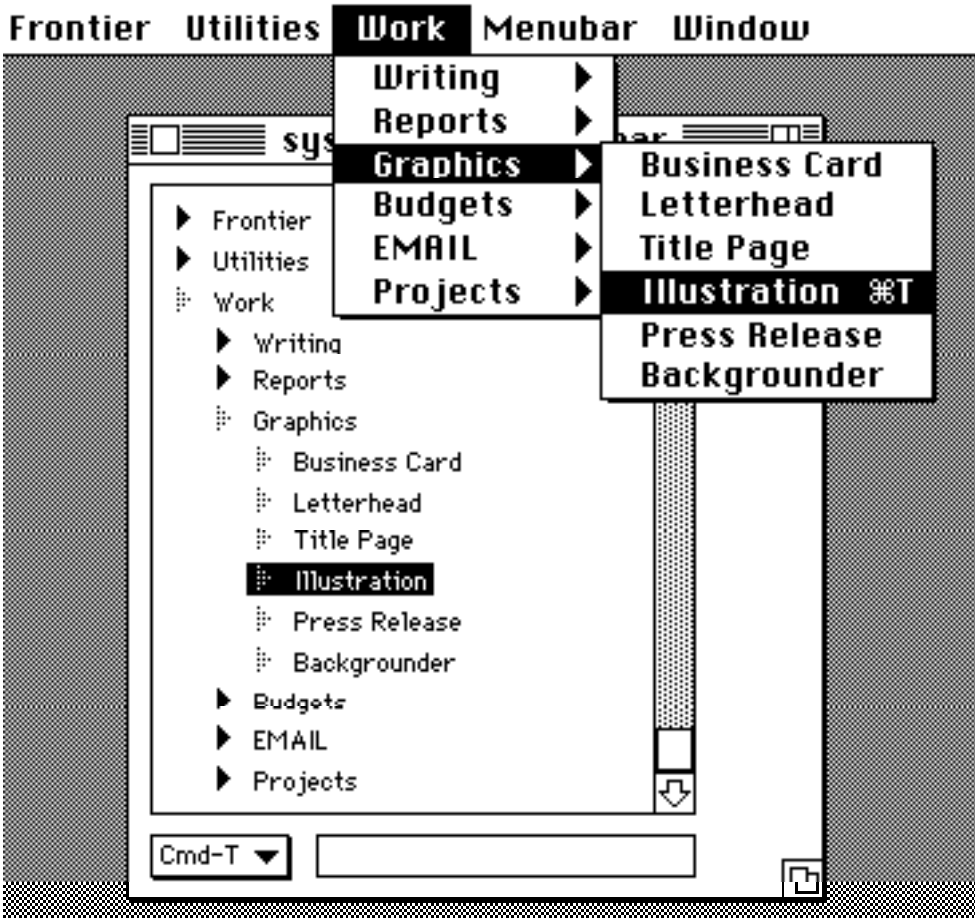
Agent scripts are used to monitor folders, especially on shared disks. A server running Frontier or Frontier Runtime can run an agent script that watches the folder. When a file appears in the special folder, the agent script processes it in some fashion.

In addition to agent scripts, there are special tables for scripts that run on startup and shutdown, or scripts that respond to incoming Apple Event messages. Even the interpreter’s runtime stack is accessible thru the object database hierarchy.

Much of the culture of Frontier is implemented as scripts stored in the object database, so you can examine and customize the scripts and add your own. When we upgrade Frontier, we only upgrade the parts of the database that we created, and leave the parts that are subject to customization untouched.

Menubar editor, Menu Sharing

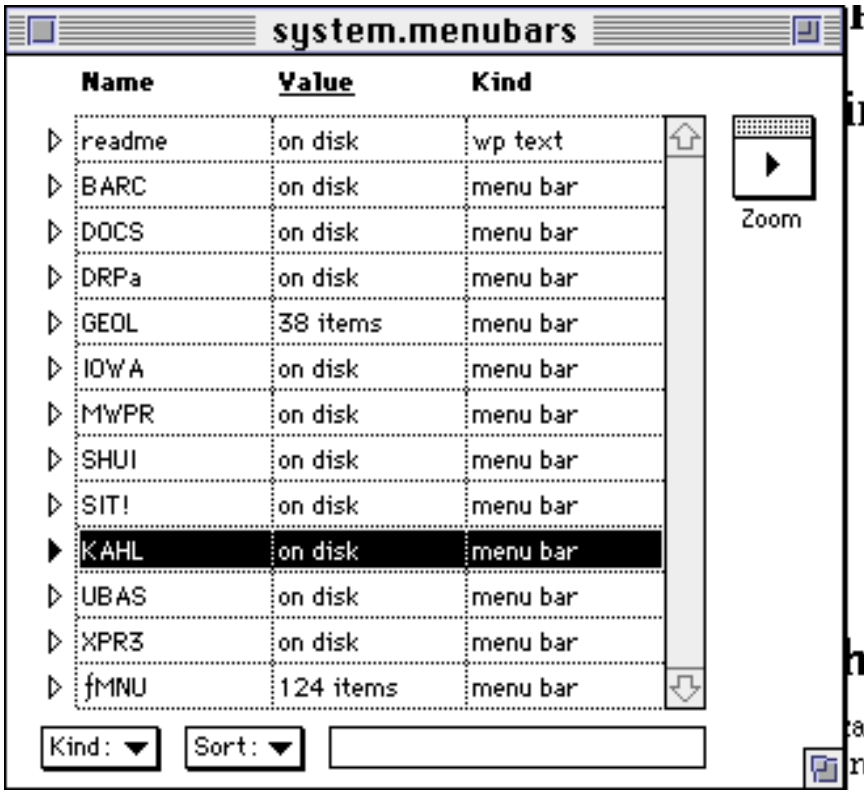
Like Frontier's script editor, the menu editor works with an outline. The outline is hot-wired to the actual Macintosh system menu bar. When you make a change in the menu editor it's immediately reflected in the system menu bar. In the following screen shot, the script writer is editing the Graphics sub-menu of the Work menu in Frontier's menu bar:



The outline hierarchy is reflected in hierarchical menus. Each main heading is a menu, each sub-head is a command or a sub-menu. To edit the script linked into each menu item, click on the Script button (it's hidden in the screen shot above). A script window opens.

This menu editor can also be used to add commands to applications that support the Apple Event-based menu sharing protocol. Examples include Think C and Symantec C++ 6.0, Quark XPress 3.2, Storm Technologies PicturePress 2.0, and StuffIt Deluxe 3.0 and StuffIt Lite 3.0 from Aladdin Systems. Thru the FinderMenu extension, included with Frontier 2.0, you can add commands to the Finder's menu bar.

To see how menu sharing works from a script writer's perspective, open the table at system.menubars:



This table associates a menubar object with a Macintosh application that supports menu sharing. KAHL contains the shared menu for Think Project Manager. XPR3 contains the shared menu for Quark XPress 3.2. When one of these applications launches, it sends a series of Apple Events to Frontier to request its shared menus. When the user selects one of the commands, the application sends a message to Frontier to run the script that the user selected.

Frontier Software Developer Kit (or Frontier SDK) includes the Menu Sharing Toolkit which makes it easy for developers to open their menubars to script writers. It can be downloaded from any of UserLand's on-line services and is included in the Frontier 2.0 package. All UserLand developer toolkits are royalty-free and provided in full C source code.

Scripts on the Desktop

Desktop Scripts

Desktop scripts are files that contain a script that runs when the file is double-clicked on. The script can be programmed to operate on the folder it was launched from.

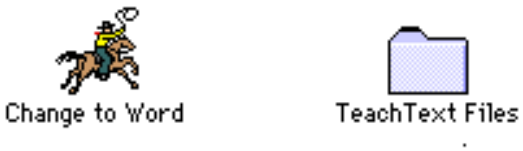
To create a new desktop script, choose the New Desktop Script command in Frontier's UserLand menu. A new script is created in the system.deskscripts table. To export it, choose the Export a Desktop Script command from the Export sub-menu of the UserLand menu.

The following desktop script searches the folder it was launched from for TeachText files. When it finds one the script changes its creator id so that Microsoft Word will open it.

```
local (f, folder)
folder = file.folderFromPath (system.deskscripts.path)
fileloop (f in folder, infinity)
    if file.creator (f) equals 'ttxt'
        file.setCreator (f, 'MSWD')
```

Before a desktop script runs, Frontier sets up system.deskscripts.path so that it contains the full path to the desktop script file. The script can loop over all files in a folder, or store information in the resource fork of the script file.

Here's what a desktop script looks like in the Finder:



If you run the script in the same folder as the TeachText Files folder, it changes all the TeachText files to Word files.

Droplet Scripts

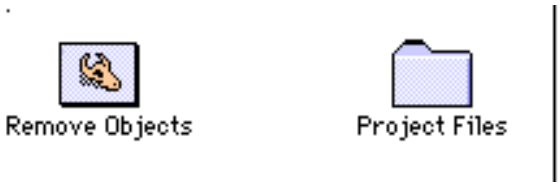
Droplets are small applications that have an embedded script that runs once for each file, folder or disk icon that’s dropped onto the application.

Frontier communicates with the droplet script through the system.droplet table. The script can also use this table to store values that persist across calls to the script. In the following example, the droplet keeps track of the amount of disk space it reclaimed in system.droplet.bytessaved.

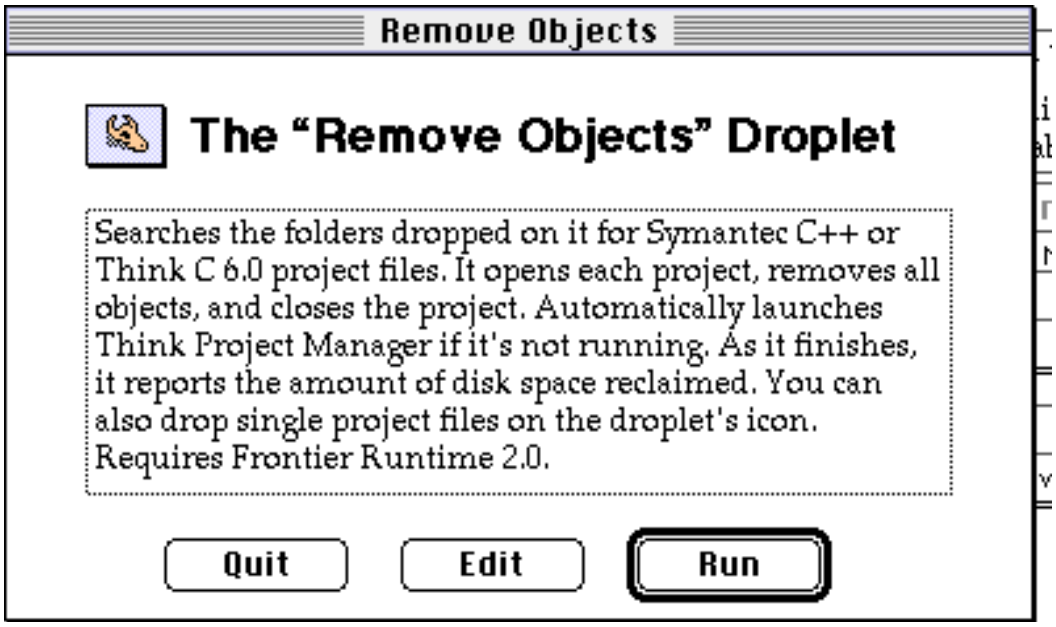
Here’s a droplet script that searches the folders dropped on it for Think C project files. It opens each project, removes all objects, and closes the project. It automatically launches Think Project Manager if it's not running. As it finishes, it reports the amount of disk space reclaimed:

```
if system.droplet.startup
    system.droplet.bytessaved = 0 «keeps track of space we reclaimed
if system.droplet.closedown
    if system.droplet.bytessaved > 0 «report to the user
        dialog.notify (string.kBytes (system.droplet.bytessaved) + " reclaimed.")
    delete (@system.droplet.bytessaved)
    return
on checkfile (f)
    if think.isProjectFile (f)
        local (origfilesize = file.size (f))
        think.openProject (f)
        think.removeObjects ()
        think.closeProject (true)
        system.droplet.bytessaved = system.droplet.bytessaved + (origfilesize - file.size (f))
if file.isFolder (system.droplet.path)
    local (f)
    fileloop (f in system.droplet.path, infinity)
        checkfile (f)
else
    checkfile (f)
```

This is what a droplet looks like on the desktop:



Drag the Project Files folder onto the Remove Objects icon. Here's what you see:

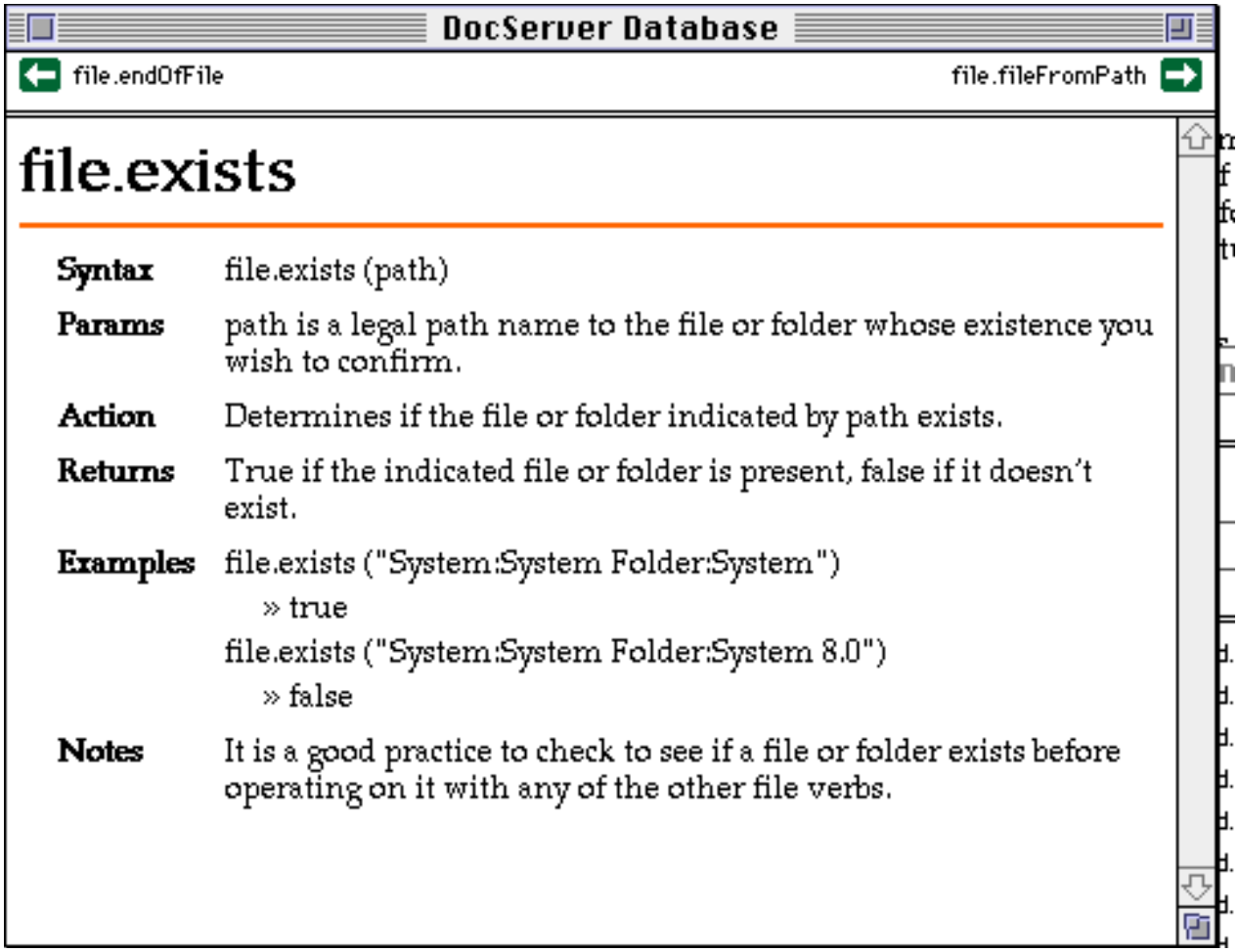


Droplets are menu-sharing-aware, so you can include configuration commands in the droplet's menu bar, implemented as scripts.

A full tutorial and reference guide, sample code, and source for the Droplet Developer Kit is available in the Extras folder of Frontier SDK.

DocServer

DocServer is the on-line documentation tool for Frontier script writers. Each “page” in the DocServer database describes one of the UserTalk language verbs. The example below shows the docs for the file.exists verb. Information on the syntax, parameters, action and returned value of the verb are included.



DocServer is connected to Frontier thru Apple Events. If you select a name in Frontier, and hold down the Control key while double-clicking on it, DocServer will display the documentation for that verb.

DocServer is an open tool. Other developers are encouraged to provide Apple Event verb documentation in DocServer format.

Architecture & Economics

Frontier scripts can run over AppleTalk-compatible networks, sending messages to other copies of Frontier or Runtime, or to other Apple Event-aware applications.

Scripts are re-usable. Scripts can call other scripts, passing parameters and receiving returned values. Script writers can add new verbs to the language.

Scripts can be compiled into an intermediate non-human-readable form. In an upcoming release of Frontier it will be possible to distribute scripts in this form.

Compiled machine code can be stored in the object database and called exactly as scripts are called. Two communication protocols are supported: the well-known HyperCard 1.0 XFCN protocol and a new and more powerful protocol known as UCMD, which is compatible with the AppleScript OSAX protocol. Toolkits for developing both kinds of code resources is included in Frontier SDK 2.0.

The language is case-insensitive. fileFromPath is the same identifier as filefrompath and FiLeFRoMPAtH.

Frontier is multi-threaded. Each script runs in its own thread, and can use the object database to communicate with each other and for synchronization.

On-line support is provided thru CompuServe, America On-Line and AppleLink. Support forums are staffed by Frontier's lead developers. Sample scripts, toolkits, and documentation, all without additional charge, can be downloaded from UserLand's on-line support services:

- CompuServe: Type GO USERLAND at any ! prompt.
- America On-Line: Keyword USERLAND.
- AppleLink: UserLand Discussions under Third Parties.

Frontier Runtime runs scripts, but has no interactive script editing or debugging features. It's available as an inexpensive shareware package, with a 30-day free trial period. Network site licenses are available for as little as $20 per machine for networks with more than 10 Runtime users. Runtime can be bundled with commercial applications that support Apple Events for $100 per year, no royalty. All prices are in U.S. dollars.

A special shareware Scripting Starter Kit (SSK) is available for Think C and Symantec C++ 6.0 developers. It can be downloaded from any of UserLand's on-line services.

Frontier is a commercial product with a suggested retail price of $249.

Both Frontier and Frontier Runtime require System 7.0 or greater. Frontier requires a minimum of 1024K, Frontier Runtime requires 512K.

Q&A

Why would I add Frontier capability over AppleScript?

Both AppleScript and Frontier build on the same protocol: Apple Events. It’s impossible to support one without supporting the other.

The only extra protocol we ask you to support is menu sharing. You should add it because script writers want it, and it’s very easy to implement.

How do I make verbs for my app?

Suppose your app, named mathApp, implements a single Apple Event that takes one parameter, a long. It multiplies it by two, and returns the result.

Create a new table for your application at system.verbs.apps.mathApp. Create a new script in that table, call it “doubler”. Here’s what the script looks like:

```
on doubler (x)
    return (appleEvent ('MIKE', 'CUST', 'DOUB', '----', long (x)))
```

It returns the value of a call to Frontier's built-in appleEvent verb. 'MIKE' is the creator id of mathApp, 'CUST' is the class of the verb, 'DOUB' is the verb identifier. The next two parameters provide the key, '----', and the value of the single long parameter to the Apple Event.

A script writer can then call your verb exactly as if it built into the language. Here's an example that displays 24 in a standard alert dialog:

```
dialog.alert (mathApp.doubler (12))
```

Full details are provided in Frontier SDK 2.0, Extras folder, Frontier Install File Creator sub-folder.

How long does it take to add menu sharing to my app?

You add calls to Menu Sharing Toolkit routines in five places:

1. **Where you initialize Macintosh toolkits. [Add a call to InitSharedMenus to set up globals and install handlers for messages sent by the menu sharing server.]**
2. **In your menu handling routine. [Call SharedMenuHit to filter out shared menu selections. If it returns false, process the menu selection normally, if it returns true, it was a shared menu command, return to your main event loop.]**
3. **In your keystroke handler. [Call CancelSharedScript if the user hits cmd-period and SharedScriptRunning returns true. Otherwise, process the keystroke as you normally would.]**
4. **In your event handler. [On receipt of a null event, call CheckSharedMenus to load the shared menus, or reload them if they changed.]**
5. **In each Apple Event handler. [Call the toolkit routine SharedScriptCancelled . If it returns true, your handler should returns noErr immediately.]**

If you're working in Think C or Symantec C++ it could take as little as 20 minutes to find these places and add the calls. There's a reasonably good chance it will work the first time, as it has for many others.

A full tutorial and reference guide, sample code, and source for the Menu Sharing Toolkit is available in the Toolkits folder of Frontier SDK.

Final Note

If you have any questions, comments or suggestions, please get in touch through one of UserLand's on-line services. If you're an AppleLink user, check out the UserLand Discussion Board under the Third Parties icon. On CompuServe, visit the UserLand Forum in the Computing Support section, or enter GO USERLAND at any ! prompt. On America On-Line, enter the keyword USERLAND.

About the Author

Dave Winer is founder and president of UserLand Software. He and Doug Baron are the lead developers at UserLand. Dave has been a commercial software developer since 1979, and shipped his first product, the ThinkTank outliner for the Apple II in 1983. The IBM PC version shipped in 1984. As founder and president of Living Videotext, Inc. he shipped one of the first Macintosh applications, ThinkTank 128, in mid-1984, and in 1986 shipped the award-winning MORE 1.0, followed by MORE 1.1c in 1987. Living Videotext merged with Symantec shortly after that and Dave moved on to start the development of Frontier. Frontier 2.0 received MacUser's Eddy award for best development tool of 1992.