Frontier Install File Creator

UserLand Software, Inc.

UserLand Software is located at 400 Seaport Court, Redwood City, CA 94063. 415-369-6600, 415-369-6618 (fax). UserLand and Frontier are trademarks of UserLand Software, Inc. Other product names may be trademarks or registered trademarks of their owners.

Email: userland.dts@applelink.apple.com. If you're an AppleLink user, check out the UserLand Discussion Board under the Third Parties icon. CompuServe users enter GO USERLAND at any ! prompt.

Comments, questions and suggestions are welcome!

Background

You've added Apple Event support to your application. You've installed the Menu Sharing Toolkit. You're almost ready to ship.

But how are script writers going to learn about your Apple Event support?

This document is part cookbook, part style guide. If you follow it carefully, Frontier script writers will love your product.

10/9/92

An Example

Check out "Minimal Applet" in the Sample Code folder. In addition to being an example of programming with the Applet Toolkit, it also has a readme file, a Frontier install file, and a DocServer text file included in the "MinApp & Frontier" sub-folder.

MinApp.Frontier is a Frontier install file. It contains an embedded script that adds support for your application to a script writer's object database.

DocServer Source Text is a text file that contains documentation for the Apple Events that your program implements.

The "MinApp & Frontier" folder illustrates the principles outlined in this document. Feel free to copy it and modify the contents so it works for your application.

10/9/92

Getting Started

Make a list

You'll need the following information before configuring Frontier for your application:

1. **Your application's name. It should be short, but not so short as to be cryptic. It must be a language identifier, and therefore should contain no spaces, and it must start with an alphabetic character. Underscores are allowed but not recommended. By convention, application names are MixedCase: like BarChart, QuickMail, PageMaker.**

2. **Your application's creator id. This is the 4-character creator id for your application and your data files.**

3. **Does your application support menu sharing? For more information, see the Menu Sharing Toolkit folder.**

4. **The application file. You may have several copies of your program on your system, Frontier needs to know which copy to launch when communicating with your application.**

Set up Frontier

First, launch Frontier.

A new version of the Commercial Developers Suite is included in this folder. Double-click on its icon to install.

Select Commercial Developers from the Suites menu. A new menu appears in Frontier's menu bar: the Glue menu. It's name has historic significance from the days when Frontier install files were called "glue files."

Select "Enter Your App's Name" from the Glue menu. A series of dialogs guide you thru the process of setting up Frontier to work with your application.

Your Glue Table

Where your glue table lives

If you entered the name "MyApp" for your application, the glue table will be created at system.verbs.apps.MyApp. Use Frontier's Jump To command (cmd-J) to open your glue table.

Frontier scripts can reference this table directly because the address of system.verbs.apps is in Frontier's "paths" table. See pp 85-86 in the Frontier User Guide for details.

MyApp.appInfo

An "appInfo" table was created too. It has several bits of information that help Frontier work with your application:

1. app1Supported **is a boolean, indicating whether or not the application supports the app1 verb set.**

2. sharedMenus **is a boolean, it indicates whether or not the application implements Frontier's menu sharing protocol.**

3. name **is a string, it's used in Frontier messages that mention your application by name. Without this string, Frontier would have to refer to BarChart as 'BARC' That's not as friendly as using a longer string. This name does not have to be an identifier, so you can change it to whatever makes sense for your application.**

4. path **is a string, set automatically and maintained by the app.start Frontier verb.**

5. id **is a 4-character string, the creator id of your application.**

MyApp.id

This is the target of all the Apple Events sent to your application. Usually it is just the creator id of your application. However, if Frontier is talking to your application over a network, it contains the network address of your application. If you're debugging with THINK C, it contains the id 'KAHL'.

You should never assign to MyApp.id, but all your glue scripts should refer to it. Details follow.

Write your first glue script

Let's write a "glue script" to connect to one of the Apple Events implemented in your program. We'll start with a very simple one that simply tells your program to quit.

Bring your glue script table to the front. It's the one named system.verbs.MyApp.

From the Table menu in Frontier, select the New Script… command. Enter the name for your script:

```
Name for new script cell?
quit
              [ Cancel ]   [[ OK ]]
```
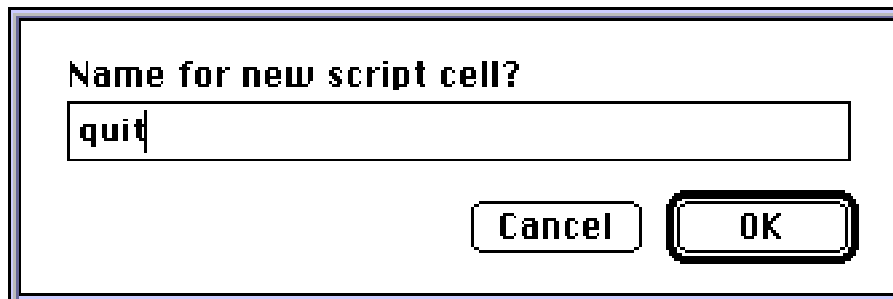
A window opens with an empty script. Enter the following into the new window (use the tab key to indent the second line):

```
system.verbs.apps.MyApp.quit
      [ Run ]  [ Debug ]  [ Compile ]

on quit ()
    return (appleEvent (My App.id, 'aevt', 'quit'))
```

Launch your application. Then bring up Frontier's Quick Script window (press cmd-semicolon) and type:

```
Quick Script

MyApp.quit ()
                                          [ ▶ ]
                                          Run

                    true
```
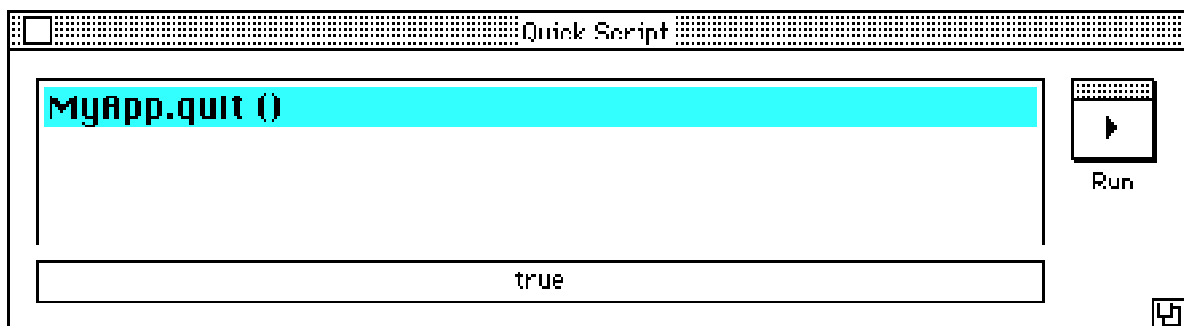
Then click on the Run button. Frontier runs the MyApp.quit script; in turn, it sends an Apple Event to your application telling it to quit. If you've correctly implemented the required event suite (fingers crossed!) your application should no longer be running.

Let's do another experiment. Bring the Quick Script window to the front and click on the Run button again. You should see an error message:

```
┌─────────────────────── Error Info ───────────────────────┐
│ □                                                        │
│  ┌──────────────────────────────────────┐   ┌────────┐   │
│  │ Can't send the message because "MyApp" isn't │   │   ▶    │   │
│  │ running or isn't IAC-aware.            │   └────────┘   │
│  │                                        │     Go To      │
│  │                                        │                │
│  │                                        │                │
│  └──────────────────────────────────────┘              ⊡  │
└──────────────────────────────────────────────────────────┘
```

If you click on the Go To button, Frontier takes you to the place that the error occurred, the call to the appleEvent built-in.

The second glue script

Now let's write a script that connects to the required 'odoc' Apple Event.

In Frontier, bring system.verbs.MyApp to the front, select New Script… from the Table menu and name the script openDocument. Then enter the following script into the new window:

```
system.verbs.apps.MyApp.openDocument
       [ Run ]  [ Debug ]  [ Compile ]

 on openDocument (filePath)
    return (appleEvent (MyApp.id, 'aevt', 'odoc', '----', alias (filePath)))
```

Launch your program, then enter the following into the Quick Script window and click on the Run button:

    MyApp.openDocument ("System:Test Document")

Inside the quotes, enter the full path to a document created by your application. Switch into your application. You should see a window displaying the contents of the document.

The remaining glue scripts

The rest of your glue scripts will follow the same pattern. Write one script to connect to each of the Apple Events supported by your application.

To see an example of a complete glue scripts table, double-click on MinApp.Frontier in the Minimal Applet folder, and jump to system.verbs.apps.MinApp.

Sample code

If you're looking for examples of how to implement the other side of these glue scripts, have a look at the "Menu Sharer" sub-folder in the Sample Code folder. It builds on the IAC Tools library to implement its five Apple Event handlers.

MyApp.readme

Create a wp text object in your glue scripts table with pointers showing script writers where to go for more information.

MyApp.examples

You should include some example scripts to show script writers how to work with your application in a sub-table of your glue script table. Call it examples.

See MinApp.examples for some ideas. They don't have to be profound, actually the simpler the better. Give us ideas on how to access the power of your program.

innerCasing

By convention, the names of glue scripts are innerCase. The first character is lower case. Each new word in the script name begins with an uppercase letter.

Your Shared Menu

Where your shared menu lives

If your application's creator id is 'MYAP', the shared menu for your application is at system.menubars.MYAP. Use Frontier's Jump To command (cmd-J) to open your menubar.

Its Purpose

When you add menu sharing to your application you're allowing script writers to add commands to your program. When preparing a sample menu bar to ship with your application, you're providing examples that will show script writers what they can do.

It's created automatically

When you entered your application's name using the Glue menu, a new shared menu is created automatically. Initially this menu has three real commands and three placeholders for demo scripts.

Conventions

The art of developing shared menus has progressed substantially. This section details some of what we've learned. It's a style guide, not hard and fast rules. But a bit of consistency between shared menus will make it easier for people to learn how to write scripts for new apps.

1. **Use hierarchies sparingly.**

   If you have ten commands to share as sample scripts, don't use any hierarchies. Keep a balanced appearance, put the same care into organizing the structure of your shared menu as you would with a "hard wired" menu.

2. **One menu, not five.**

   Leave lots of room for script writers to be creative. The more menus you write, the less inviting for the script writer. Choose your examples carefully.

3. **Call it the Scripts menu.**

   Give it a name to set it off from the rest of your menus. The script writer can change this later.

4. **Include an About command.**

It should be the first item in the menu. It should bring Frontier to the front, use the built-in dialog.notify command to display a short message and then bring your application to the front after the user clicks on OK.

5. **Cmd-M opens the shared menu.**

This should be the second-to-last item in your shared menu. It opens the shared menubar for the script writer to edit it. If cmd-M is already in use in your program, leave the cmd-key blank in the menu bar editor.

6. **Cmd-T opens the scripts table.**

This should be the last item in your shared menu. It opens your application's scripts table. If cmd-T is already in use in your program, leave the cmd-key blank in the menu bar editor.

When your menu bar is created initially it is set up automatically to follow these conventions. Of course you can and should edit it.

Exporting your Install File

When to do it

In the process of releasing Apple Event support for your application, you will export your install file many times. Basically, any time you want to give a copy of your program to a script writer, you should export a current install file and include it with your application.

How to do it

It's very simple. Select Commercial Developers from the Suites menu to add the Glue menu to the menu bar. Be sure that you've entered the name of your application. Then choose Export Install File… from the Glue menu.

DocServer Docs

DocServer

UserLand includes a copy of DocServer with Frontier, and it is available for downloading on the AppleLink and CompuServe on-line services. The purpose of DocServer is to provide a quick and simple way for script writers to access documentation for your glue scripts.

commercial.sampleDocs

Preparing your DocServer docs is done in Frontier, using the built-in outliner.

Open the outline at commercial.sampleDocs. As you browse thru it you'll see that there's a common structure. Each verb has information about its syntax, parameters, action and returned values. Also included are example script fragments and often notes and pointers to other relevent verbs.

To prepare documentation for your verbs, edit this outline. Then select the Compile DocServer Text command to export the outline into a format that can be read by DocServer. To see what your docs look like, launch DocServer and doube-click on the file.

Include this file in your "MyApp & Frontier" folder.

MyApp & Frontier

The Folder

You should prepare a folder called "MyApp & Frontier." You may choose to include it on your shipping disk or upload it to UserLand's CompuServe forum (Library 3, Scriptable Apps). It should contain three files:

Read Me

You should include a Read Me file in TeachText format. Tell the reader how to install support for your application in Frontier. Keep it direct and simple. You're talking to script writers, they're more technical than the average end user. People will tune out if the installation instructions are too complicated.

MyApp.Frontier

This is your Frontier install file. This file is created by the Export Install File… command in the Commercial Developer Suite, included in this folder. Details are provided in a separate section.

MyApp.DocServer

This is a text file, produced by the Compile DocServer Text command in the Commercial Developers Suite. It's file type is TEXT, the creator is DOCS. When the script writer double-clicks on this file, UserLand's DocServer app launches and loads the text into its database.

'aete' Resources

Jump-starting your glue table

Terminology or 'aete' resources are the equivalent of Frontier install files for Apple Computer's AppleScript software. If you already have such a resource for your application, you can use the Commercial suite's "Import 'aete' Info" command to jump-start your glue table. The command scans your application's terminology resource and creates a set of glue scripts for the events you support. It also adds values to your table corresponding to any classes, properties, and enumerators you've defined.

All of the items are named according to the terminology in the resource, with multiple-word names mapped to innerCase identifiers. Classes and properties are represented as string4Type values, while enumerators are binaryType [enum] values.

If you fully support the Core or Miscellaneous Standards suites, a set of scripts is created that call Frontier's corresponding core or misc verbs. So in cases where Frontier defines more than one verb for an event, such as core.get and core.getAs, your table will include matching entries. For individual suite events and custom events, a glue script is created directly from the 'aete' data.

Hand-tuning the result

The resulting glue table will be almost ready for prime time. But it should be reviewed carefully, and fine tuned as necessary. In particular, the 'aete' resource doesn't provide a name for an event's "direct" parameter. Frontier makes a best-guess at a name based on the parameter's datatype, but it's likely that you can come up with something more meaningful. For other parameters, the name retrieved from the resource may be a UserTalk keyword; check for this by making sure that every glue script compiles.

As you go through the hand-tuning process, you may want to take notes about the changes you make. If you end up wanting to re-import from the resource later, you'll need to apply the changes again.

Once you have the glue scripts tuned to your satisfaction, you should think about providing higher-level verbs that build on this basic set. Consider the types of tasks that script writers might want to accomplish with your application; does it make sense to bundle several common steps into a single verb? For object model applications, there may be events that are likely to be called with a specific set of parameters. For example, a charting application might add a "createNewChart" verb that simplifies the use of the core "create" event. The time you spend smoothing out the scripting interface here will make a lot of difference to script writers.

Scriptability Checklist

Launching in the background

Many apps assume that the only way they can be launched is by the user double-clicking on something in the Finder, or selecting their application from the Apple menu. In other words, they assume that when they are launched they will be the frontmost application.

However, when a script launches your application it will not be the frontmost application until the script brings you to front. If you are performing a server function for the script, you may never be brought to front.

Therefore, techniques like splash screens work only if they do not require interaction from the user. In fact, it would be more polite to only show a splash screen if you are the frontmost app.

Some early System 7 apps break this rule, and as a result are much less useful to script writers.