

Menu Sharing Toolkit 2.0

UserLand Software, Inc.

© copyright 1992, UserLand Software, Inc.

UserLand Software is located at 400 Seaport Court, Redwood City, CA 94063. 415-369-6600, 415-369-6618 (fax). UserLand, Frontier and Frontier Runtime 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

Frontier's Editable Menu Bar

One of Frontier's many capabilities is that it gives the script writer an easy way to edit the contents of its own menu bar. When the user selects an item from the menu, Frontier runs the script that's linked into the menu item.

Thru the menu sharing protocol, implemented by the Menu Sharing Toolkit, that capability is extended to allow Frontier to add new hierarchic structures of shared menus to any Macintosh application that includes the Toolkit.

Menus sharing adds a set of standard Macintosh menus at the end of your application's menu bar. Script writers can add commands to these menus using Frontier. Any changes to your menus are automatically visible as soon as the script writer switches back to your application.

Menu sharing does not in any way alter the performance of your menu commands, it just adds new commands to your menu bar.

Applications-as-Development-Platforms

Menu sharing allows script writers to view your application as a customizable development platform. They can add commands that automate your application just as if it had an integrated scripting language. Even if your application already has an integrated scripter, it makes sense to support the menu sharing protocol because scripts can easily launch other applications and integrate their capabilities with your program and the Macintosh operating system.

Script users, usually less technical people than script writers, will see simpler commands in these menus. Examples include: prepare for a meeting, send a message to everyone working on a specific project; write a press release; or hire a new employee. There are as many potential custom scripts as there are Macintosh users. They will only be aware that these commands were “put into” your application by a friend they work with, or their organization’s network manager. The details of this technology are completely open to the script writer, but neatly hidden from the end-user.

To many potential users of your product, commands like New, Cut, Paste, and Zoom are like assembly language. By opening your product to menu sharing you offer more technical users an easy way to simplify your product for less technical users. This means your technology is more useful to a much larger number of people, and can only contribute to making your product more competitive and more profitable.

The Menu Sharing Toolkit

In order to be menu sharing-aware, an application includes the Menu Sharing Toolkit and inserts calls to Toolkit routines in several places: on initialization; in the main event loop; before processing a menu selection; where the user presses cmd-period; and in each Apple Event “handler” routine.

No Royalty or License Fee

There is no royalty or license fee for use of the Menu Sharing Toolkit. It’s provided in source form so that developers know exactly what the Toolkit is doing on their behalf, and so it can easily be adapted to different development environments.

You may include this code in any Macintosh application, but you must maintain our copyright notice at the head of each Menu Sharing source file.

Menu Sharing is Open

We're publishing source code for the client side of the menu sharing protocol, and plan to support menu sharing in all future versions of UserLand Frontier and all other software products developed by or published by UserLand Software.

We believe menu sharing should be an open specification, with freely distributed source code. Everyone wins if it's widely supported on both sides of the equation, by clients (the vast majority of "sharing-aware" programs) and by servers (Frontier, and competitive products).

Icing on the Cake

It makes little or no sense to add menu sharing if your application is only minimally IAC-aware. Scripts launched from your menu bar must be able to call back into your program to open or close a window, get a value, add a record, dial the phone -- basically to do the types of things your program was designed to do.

Menu sharing, therefore, is like icing on a cake. Once you've added a reasonably complete set of scriptable IAC messages, the next step is to open up your menu bar to allow script writers to insert new commands that are implemented in scripts.

Cookbook

Introduction

In this section we'll step through the source code for "MenuSharer," a very minimal Macintosh program, whose main reason for existing is this tutorial. It pays to browse through the MenuSharer source code before installing menu sharing in your program. MenuSharer is written in THINK C 5.0. To view the source code, open `menusharer.1` with THINK C.

The Toolkit is implemented in `menusharing.c`. Its header file is `menusharing.h`. To see how these routines are called, page through `main.c`. You'll see calls to Toolkit routines in five places:

1. In `handlemenu` -- we call `SharedMenuHit` to run scripts selected from a shared menu.
2. In `handlekeystroke` -- we call `SharedScriptRunning` and `CancelSharedScript` if the user had pressed `cmd-period` to cancel a running script.
3. In `handleevent` -- we call `CheckSharedMenus` to reload the shared menus if they have been changed by a script writer.
4. In `main` -- we call `InitSharedMenus` to set up globals and install handlers for IAC messages that may be sent by the menu server.

If you have Frontier handy, double-click on MenuSharer's Frontier install file, and edit the menu bar at `system.menubars.MENS`.

It may take a couple of readings of this section, browsing the source code, and reading the reference section for the whole concept to "click" -- but please stay with it -- menu sharing is really very simple, elegant, and powerful.

Following is a step by step "cookbook" for adding menu sharing to any Macintosh application.

Step #1 -- Initialize the Toolkit

After initializing the Macintosh using `InitGraf`, `InitFonts`, etc., call the Toolkit `InitSharedMenus` routine:

```
if (!InitSharedMenus ()) /*failed to initialize*/
    goto error;
```

This call initializes menu sharing globals, and installs Apple Event handlers for the two IAC messages it must be prepared to receive from Frontier. `InitSharedMenus` returns false if the initialization failed for some reason. Your application may want to continue, or as MenuSharer does, simply display an alert and exit to the operating system.

Step #2 -- Add a Call in Your Main Event Loop

In your main event loop, when you receive a null event and are not suspended, insert a call to `CheckSharedMenus`. You can keep track of whether or not you are active by setting a global flag each time you receive a suspend/resume event. It's important that you only call

CheckSharedMenus when you're active to ensure that the menubar can be updated properly.

If your menus need updating, CheckSharedMenus automatically disposes your out-of-date shared menus, and conducts a short Apple Event conversation with the menu sharing server to get your updated menus.

CheckSharedMenus takes one parameter -- the resource id for the first shared menu. If CheckSharedMenus loads new menus, they are assigned resource ids starting with this number. It must be less than 255 to allow for hierarchic menus, and must be small enough so that no menu has an id of greater than 255. If there are n menus, and the parameter to CheckSharedMenus is x , then the menu ids will range from x to $x + n - 1$.

MenuSharer calls CheckSharedMenus in its handleevent routine in main.c.

Step #3 -- Add a Call in Your Menu Handler

In your menu handling routine, insert a call to SharedMenuHit:

```
if (SharedMenuHit (idmenu, iditem)) /*it was a shared menu item*/  
    goto exit;
```

SharedMenuHit checks to see if the indicated menu is a shared menu. If so, it builds an Apple Event message instructing the server to run the script, then disables the shared menus and returns true. Your program should immediately return to its main event loop.

If SharedMenuHit returns false, the menu is not a shared menu, and you should process the command the way you normally would. menu sharing does not in any way alter the performance of your menu commands, it just adds new commands to your menu bar.

MenuSharer calls SharedMenuHit in its handlemenu routine in main.c.

Step #4 -- Add code to Your Keystroke Handler

Now that we've given the user a way to launch a script from within your application's menu bar, we have to provide a way to terminate or cancel a script. Following the standard Macintosh user interface conventions, this should be accomplished by pressing cmd-period while a script is running. The user will be able to tell that a script is running because the shared menus are disabled while a script is running.

Use `SharedScriptRunning` to determine if a script is currently running. If it returns true, and the user has pressed cmd-period, call `CancelSharedScript` to terminate the script.

Here's the `MenuSharer` code, in `main.c`, that implements script termination:

```
handlekeystroke (ev) EventRecord *ev; {
    register char ch = (*ev).message & charCodeMask;
    if (SharedScriptRunning ()) /*cmd-period terminates the script*/
        if (((*ev).modifiers & cmdKey) && (ch == '.')) {
            CancelSharedScript (); /*cancel the shared menu script*/
            return;
        }
    handlemenu (MenuKey (ch)); /*process the normal way*/
} /*handlekeystroke*/
```

Step #5 -- Add code to each Apple Event Handler

In Step #4 above, we added a call to `CancelSharedScript`. `CancelSharedScript` doesn't send an IAC message to the scripting system, it just sets a global flag indicating that the script has been cancelled. The scripting system doesn't actually find out that the script has been cancelled until it attempts to send an IAC message to the application. You could watch the flag yourself, and return an error to the caller; or you could call `SharedScriptCancelled`, which does this for you transparently.

In every Apple event handler that could reasonably be called from Frontier or a similar scripting system, insert a call to `SharedScriptCancelled`, passing it a pointer to the Apple event record and the reply record:

```
if (SharedScriptCancelled (event, reply))
    return (noErr);
```

If the script has been cancelled, `SharedScriptCancelled` replies with an error. The error string (key 'errs') is the empty string; the error number (key 'errn') is 6. By convention, the scripting system will not display an error dialog on this error, since it was initiated by the user pressing cmd-period.

If `SharedScriptCancelled` returns true, you should not process the event. Simply return `noErr` to the Apple Event Manager.

Here's what the `MenuSharer`'s `setmessageverb` Apple event handler looks like:

```

pascal OSErr setmessageverb (event, reply, refcon) AppleEvent *event,
*reply; long refcon; {
    Str255 s;
    if (SharedScriptCancelled (event, reply)) /*it was cancelled*/
        return (noErr);
    if (!IACgetstringparam (event, '----', s))
        return (false);
    IACreturnboolean (reply, setwindowmessage (s));
    return (noErr);
} /*setmessageverb*/

```

IACgetstringparam and IACreturnboolean are implemented in menusharing.c. You can call any of the “IAC” routines implemented there. Function prototypes are provided in iac.h.

Step #6 -- Include "menusharing.h"

Add a #include at the top of every file that calls menu sharing Toolkit routines, as detailed in Steps 1-5 above.

Reference Information

Introduction

The Menu Sharing Toolkit is implemented in a single C source file, `menusharing.c`. To use the Toolkit, include the `menusharing.h` header file at the beginning of every source file that calls a Toolkit routine or references a menu sharing global.

There are six menu sharing routines that your program is likely to call. Those routines are documented here. You can browse the source code to see how these routines are implemented. You may need or want to build other routines out of `menusharing.c` routines if you want shared menus to appear as popup menus, or if you're implementing shared menus in a Control Panel, XCMD or Finder extension.

In this section we try to avoid referring to Frontier by name, rather we talk about the “menu sharing server” or “menu server” application. For now that's Frontier. As we mentioned earlier, the menu sharing protocol is an open protocol, so it's possible to imagine other software playing the role of menu server.

Much of the information presented here is provided in more detail in the Cookbook Section, following. This reference section is provided primarily as a summary of the information presented in that section.

Menu Sharing Globals

All the globals needed by the Menu Sharing Toolkit are stored in a single C structure named **MSglobals**. Toolkit routines are provided to initialize and operate on the fields of this structure, but we provide the details and document it so you can monitor the performance of the Toolkit using a source-level debugger.

Here's what the declaration of **MSglobals** looks like:

```
typedef struct tyMSglobals {
    OSType serverid;
    OSType clientid;
    hdlmenuarray hsharedmenus;
    Boolean fldirtysharedmenus;
    Boolean flscriptcancelled;
    Boolean flscriptrunning;
    tyMScallback scriptcompletedcallback;
    long idscript;
} tyMSglobals;

extern tyMSglobals MSGlobals;
```

MSglobals.serverid is the 4-character identifier of the menu sharing server. It's initialized by **InitSharedMenus** to 'LAND' -- the 4-character creator id of UserLand Frontier.

MSglobals.clientid is the 4-character identifier of your program. It's set automatically by **InitSharedMenus** using the Macintosh Process Manager (see IM-VI, Chapter 29). This information is passed to menu server in an IAC message to determine the set of shared menus being requested.

MSglobals.hsharedmenus is the data structure that holds the shared menus. It's set to nil by **InitSharedMenus**. See the declaration of **hdlmenuarray** in **menusharing.h** for further information.

MSglobals.fldirtysharedmenus is a boolean. If true, the shared menus are re-loaded the next time your application becomes the frontmost process. It's set true by **InitSharedMenus** to force an update the first time thru the main event loop, and when the 'updm' IAC message is received from the menu sharing server when the script writer makes an editing change to your shared menu.

MSglobals.flscriptcancelled is a boolean. If true, the currently running script has been terminated by a call to **CancelSharedScript**, probably in response to the user pressing cmd-period.

MSglobals.flscriptrunning is a boolean. If true, a script is running and the shared menus are disabled. It's set true by **SharedMenuHit** and cleared when the menu server sends a 'done' IAC message.

MSglobals.scriptcompletedcallback is a callback routine -- if it's not nil, this routine is called

when the menu sharing server sends a 'done' IAC message. You can use this capability to trigger a screen update, or to recalculate your scroll bars after a script has completed.

Msglobals.idscript is the menu sharing server's identifier for the currently running script. When the Toolkit sends a message to the server of the script it identifies the script by passing this value to the server.

Menu Sharing Routines

Boolean InitSharedMenus (void);

Initializes the Menu Sharing Toolkit.

Establishes initial values for the fields of MSglobals and installs two Apple event message handlers -- one to catch the “menu needs update” message, and another to handle the “script has completed” message.

Returns true if the program is running on a version of the Macintosh OS that supports Apple events and if it was able to install handlers for the two Apple Events, false otherwise.

Boolean CheckSharedMenus (short);

Call CheckSharedMenus from your main event loop when you receive a null event and are not suspended via suspend/resume events. If the shared menus need updating, the Toolkit sends an Apple Event message to the server asking for your shared menus.

The MenuHandles are assigned resource ids starting with the id indicated by the single parameter. This number must be less than 255 to allow for hierarchic menus, and must be small enough so that no menu has an id of greater than 255.

CheckSharedMenus will only request menus if the program is the frontmost process. This avoids delays while the script writer is editing the shared menu. Furthermore, background updating is unnecessary because the shared menus are only visible when the application is in front.

Returns false if there was an error loading the shared menus.

Boolean SharedMenuHit (short, short);

Call this routine when you’ve received a menu selection. The first parameter is the id of the selected menu, the second parameter is the id of the selected menu item.

SharedMenuHit returns false if the menu isn’t a shared menu. Your program should process the menu selection as it normally would.

If it is a shared menu, SharedMenuHit sends an Apple Event message to the menu sharing server, requesting that the script linked into that menu item be run. Shared menus are disabled while the script is running.

Boolean SharedScriptRunning (void);

Returns true if a shared script is currently running, false otherwise.

Use this routine to determine if a cmd-period should be applied to terminating the current script using CancelSharedScript.

Boolean CancelSharedScript (void);

Call CancelSharedScript when the user presses cmd-period or otherwise indicates that he or she wants the currently running script to be terminated.

Boolean SharedScriptCancelled (AppleEvent*, AppleEvent*);

Call SharedScriptCancelled from any Apple event handler that could conceivably be called from a script managed by a menu sharing server.

If the script was cancelled by the user, SharedScriptCancelled returns a specific error code to the server, indicating that the script should be cancelled, and no error dialog should be displayed.

Returns true if the script was cancelled. If so, you should immediately return noErr to the Apple Event Manager. If SharedScriptCancelled returns false, your handler should process the event as it normally would.

Messages that the server sends

The server sends two Apple Event messages as its part of the menu sharing protocol. Both messages are handled automatically by Apple Event handlers installed by InitSharedMenus. The class for both events is the same as the 4-character creator identifier of your application.

1. When your menu has been edited and requires updating, the menu server sends an 'updm' message. The handler for this message sets an MSglobals flag true. When your program becomes the frontmost application, CheckSharedMenus sends a series of Apple Event messages to the server requesting the new menus.
2. When a shared script has completed, the server sends a 'done' message. The handler for this message simply records the fact in MSglobals and re-enables the shared menus.

Where your shared menu lives

Suppose your application's creator id is 'WXYZ'. If your application has a shared menu, it's stored in Frontier's object database at system.menubars.WXYZ.

Since MenuSharer's id is 'MENS', its menus are located at system.menubars.MENS.

If you implement menu sharing, you should also develop a sample shared menu for distribution with your product as part of your Frontier install file. The "Export Install File" command in Frontier's Commercial Developers suite automatically includes your shared menu in the file it creates if one is present in system.menubars.

Change Notes

No impact on the API

We've made some major improvements in the efficiency of the Toolkit in version 2.0. However, these changes have no impact on the API, if you built on the 1.0 API you can simply replace the 1.0 library with the 2.0 library and rebuild your application.

Further, if you installed Menu Sharing 1.0 in a shipping application, Frontier 2.0 is compatible with it. The net effect of installing Menu Sharing 2.0 is an increase in performance for script writers and users.

Where to call CheckSharedMenus

We've changed our recommendation on where to call CheckSharedMenus in your program. In SDK 1.0 we recommended calling it in your main event loop, at the same time you adjust your cursor and enable/disable menu items. In 2.0, we go more conservative, and ask that you only call CheckSharedMenus when you are not suspended and you receive a null event (WaitNextEvent returns false). See the MenuSharer program for an example.

It's much faster

We use the new support for system event handlers in the IAC Tools library. If the server app has installed a set of system event handlers to implement the server side of the menu sharing protocol, we use them instead of sending Apple Events thru the operating system. The net result is virtually instantaneous loading and updating of shared menus. This performance increase is completely transparent to your program.

Compatibility

Frontier Runtime 1.0 supports the faster form of menu sharing. Frontier 1.0 does not. A version of Frontier supporting the faster protocol is in development. This Toolkit is fully compatible with present and future versions of Frontier.

UserLand will continue to support the 1.0 form of menu sharing in all future versions of Frontier and other products. If you've shipped a product with the 1.0 Menu Sharing Toolkit, UserLand will remain compatible with it.

Sends 'kill' message when script is cancelled

We've enhanced the protocol so that someday it may not be necessary to call `SharedScriptCancelled` in each of your Apple Event handlers. Instead of just setting a flag, `CancelSharedScript` now sends a 'kill' message to the menu sharing server (Frontier) to kill the script that was launched by calling `RunSharedMenuItem`, if the server has such a handler installed. If not (Frontier 1.0 does not), it depends on the `SharedScriptCancelled` method.

However, this protocol is not understood by Frontier 1.0, so we had to leave in the old method. The Menu Sharing Toolkit knows which version of Frontier it's talking to. If it's 1.0, it simply sets the flag and returns. If it's 2.0 or greater, it sends the 'kill' message. The net: you have to leave in your `CancelSharedScript` calls in your AE handlers for now. Once we have most of the 1.0 base upgraded to 2.0, it will be possible to remove those calls.

In `RunSharedMenuItem`

We check to see if the server is still running, if not, we remove the shared menus and return false.