10/9/92

UCMDs & XCMDs 2.0

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

Frontier 2.0 supports two kinds of code extensions: UCMDs and XCMDs.

In both models, you use a C or Pascal compiler to create a code resource in a Macintosh file, and then use a Frontier "droplet" to copy the code resource into the Frontier object database. Once the code is in the database, your scripts can call them, send parameters, and receive returned values. Several example code extensions are included in this folder.

XCMD support in Frontier allows script writers to tap into part of the large base of HyperCard 1.0-compatible code extensions. The architecture of XCMDs is well-understood, and for many programmers it represents a very easy way to extend the UserTalk language.

But we didn't want to stop there. We felt that there were limits to XCMDs that could easily be erased, and there were good reasons for us to invest in a new design for code extensions, one that is very compatible with the System 7 Apple Event Manager. Those are UCMDs. The "U" is for "UserTalk," the scripting language that forms the heart of Frontier.

This folder, which is part of Frontier SDK 2.0, contains a set of tools, sample code and projects that can help get you started writing XCMDs and UCMDs that work with the upcoming release of Frontier.

System.extensions is a new standard table in Frontier 2.0. All UserLand-supplied code extensions will be loaded into this table. You can load things into this table too, in fact we suggest that you do. It makes sense to keep all code extensions in one place in the user's object database.

As usual, the sample code is provided for THINK C 5.0, but the techniques can be adapted for use with any Macintosh development system that can produce code resources.

As you browse thru the folders in the UCMDs & XCMDs folder, you'll see Frontier install files for each of the extensions. To install one in your object database just double-click on its icon in the Finder.

UCMDs or XCMDs?

Tradeoffs

If you're writing a new code extension for Frontier, you can write either a UCMD or an XCMD. Naturally, UCMDs are a better fit for Frontier. But XCMDs can be faster and smaller, and a lot of people already know how to write XCMDs. In this section we list the advantages of each kind of code extension.

Advantages of XCMDs:

- XCMDs can be faster, ranging from 1.3 times faster to 1.9 times faster. See the benchmark script at system.extensions.stringOps.test2 for background.
- XCMDs are usually smaller. The minimum code size for a UCMD is approximately 2500 bytes, XCMDs can be as small as 500 bytes.
- XCMDs are a widely understood kind of code extension. For many people there's nothing new to learn.

Advantages of UCMDs:

- One code resource can implement many functions. See Trig for an example.
- Parameters and returned values can be of a many different types, unlike XCMDs which view all data as strings.
- They can return multiple values, and scripts can access these cleanly using the complexEvent interface.
- They can have globals used to communicate between functions contained within the UCMD.
- They use the IAC Tools library, without modifications, so code can move easily from applications to code extensions and back very easily.
- Once you learn how to write a UCMD, you've learned how to program Apple Events. Since this is the form of IAC that both UserLand and Apple are evangelizing, this feature has tremendous appeal.

UCMDs

The UCMD Toolkit

A small toolkit is included in this package called the UCMD Toolkit.

The main () routine for your code extension is in ucmd.c. It sets up your globals, copies the event and reply pointers into the IACglobals record, and then calls your routine UCMDmain.

An additional routine, runscript, is provided in ucmdrunscript.c. If your UCMD wants to call back to Frontier, include this file in your project. An example is included in the scriptRunner UCMD.

Over time, the UCMD Toolkit will certainly grow in size. If you have any suggestions for services that the UCMD Toolkit could provide, please let us know.

Your UCMD main file should include ucmd.h. It contains prototypes for the two routines supplied in the UCMD Toolkit folder, and also brings in the header file for the IAC Tools library.

UCMDs and IAC Tools

UCMDs are built on top of the IAC Tools library. Each project file should include as many of the iacxxx.c files as necessary to build the project. In our sample code we include all the IAC Tools files; we depend on THINK C's smart linking to exclude the code for modules that we don't call.

When it's time to release your UCMD, you might want to copy the IAC Tools routines you depend on into a separate file to make the resulting UCMD as small as possible.

Using a compiled IAC Tools Library

If you're using a compiled form of IAC Tools as a library in your project, make sure it's built with A4 addressing, not A5 addressing. One of our test sites mistakenly built a UCMD with A5 addressing. Things didn't work too well!

There is no specific switch to turn on A4 addressing in THINK C. In the Set Project Type dialog, be sure that "Code Resource" is checked off before building the IAC Tools library.

The Apple Event refcon

Each Apple Event comes with three parameters: a pointer to the incoming Apple Event record, a pointer to the reply record, and a 4-byte refcon. In Frontier 2.0, this refcon will always be 0. It's reserved for future versions of Frontier. If you have any ideas for how we should use this refcon, please let us know.

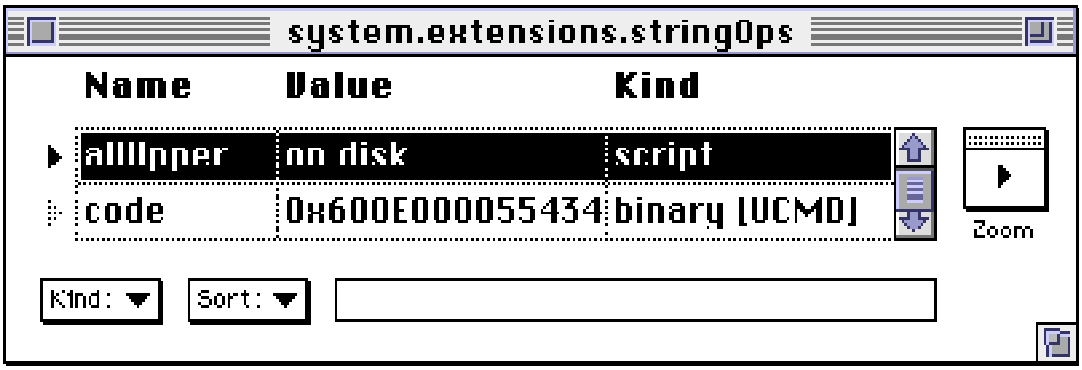
Writing a new UCMD

1. In the Finder, copy one of the folders in the UCMD folder using the Finder's Duplicate command. Change the name of the project file and the C source file to reflect the name of your UCMD.
2. Open the project file. Choose the Set Project Type command in THINK C's Project menu. Change the name item to reflect the name of your UCMD.
3. When you're ready to build the project, select the Build Code Resource command from the Project menu. Name the file xxx.ucmd, where xxx is the name of your UCMD.
4. To load the UCMD into your Frontier object database, drag and drop the xxx.ucmd file onto the Load CMD Droplet.

UCMD glue scripts

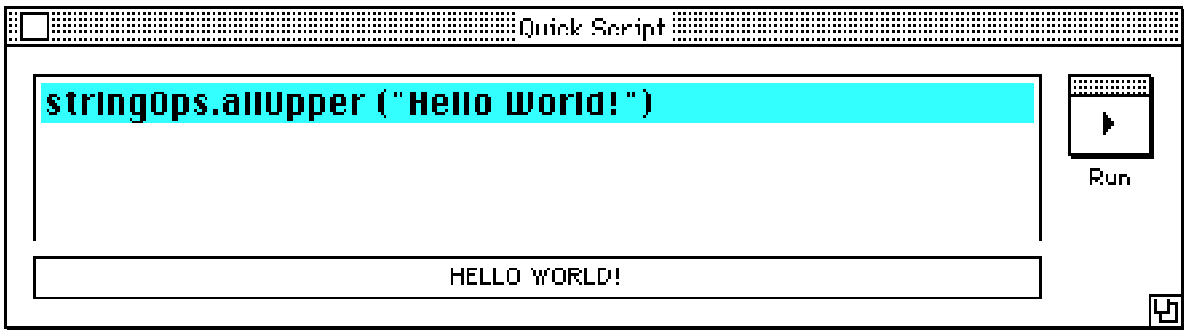
The glue scripts for UCMDs work much the same way as glue scripts for Apple Event-aware programs, with one important exception. Instead of the target of the appleEvent call being the creator id of a program, the target is the address of the object database cell that holds the UCMD code.

The stringOps UCMD implements a single Apple Event, it converts its string parameter to upper case. There are two entries in the system.extensions.stringOps table, allUpper and code.



Here's how the allUpper script connects outside callers to the UCMD:

You can call it from the Quick Script window as follows:



By convention, all UCMD and XCMD code objects are loaded into a sub-table in system.extensions with the same name as the code resource. Inside the sub-table, the name of the XCMD or UCMD is “code.” For UCMDs there are one or more glue scripts that call the appleEvent built-in to talk to the UCMD code. The only difference for XMCDs is that their single glue script uses the callXCMD Frontier kernel call.

These conventions are supported by the Load CMD droplet.

XCMDs

XCMDs versus XFCNs

We use the term “XCMD” to refer to all X-things, as seems to be the convention in the Hyper world.

Why use XCMDs instead of XFCNs? There doesn’t seem to be a good reason. All Frontier scripts are expected to return a value, even if it’s just a boolean true. Therefore, if you’re writing a new XCMD code extension, we suggest that you return a value, and use the XFCN shells we provide as a starting point.

XCMD callbacks

HyperCard 1.0 callbacks are supported to the extent that they make sense in the Frontier environment, including most of the HyperTalk Utilities and all of the String Utilities and String Conversions. In addition, the HyperCard 2.0 SendHCEvent callback is supported.

RunHandler and the Field Utilities are not currently supported. Unsupported callbacks are well-behaved; they return NIL and set the return code to xresNotImp.

When calling EvalExpr, SendCardMessage, or SendHCMessage, the message or expression must be a valid UserTalk script. All three of these callbacks execute the script, but only EvalExpr returns the result.

When getting or setting a global, any value in the current script’s scope can be named. If SetGlobal is called with a name that isn’t defined and isn’t a dotted database path, the value is placed in the scratchpad table. If GetGlobal is called with a name that isn’t defined, an error is returned.

In general, we do not recommend that you call WaitNextEvent in your XCMD, but if you do it’s important that you use the SendHCEvent callback to allow Frontier to processes events that you don’t respond to.

Writing a new XCMD

1. In the Finder, copy one of the folders in the XCMD folder using the Finder's Duplicate command. Change the name of the project file and the C source file to reflect the name of your XCMD.
2. Open the project file. Choose the Set Project Type command in THINK C's Project menu. Change the name item to reflect the name of your XCMD.
3. When you're ready to build the project, select the Build Code Resource command from the Project menu. Name the file xxx.xcmd, where xxx is the name of your XCMD.
4. To load the XCMD into your Frontier object database, drag and drop the xxx.xcmd file onto the Load CMD Droplet.

The callXCMD built-in

The callXCMD verb returns the value returned by the XCMD or XFCN — a string. If an XCMD doesn't return a value, callXCMD will return true. If an XFCN doesn't return a value, callXCMD returns the empty string.

The first parameter to callXCMD is the address of the XCMD binary object. It is followed by the XCMD's parameters, all of which are implicitly coerced to strings.

HyperCard 1.0 callbacks are supported to the extent that they apply to the Frontier environment. This includes EvalExpr, SendCardMessage, SendHCMessage, the memory and string utilities, and the string conversions. RunHandler and the field utilities are not currently supported. Unsupported callbacks are well-behaved; that is, they have no effect or return null values as appropriate.

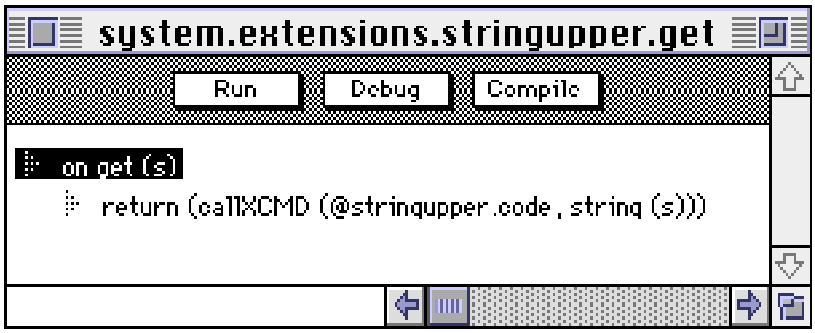
Using the GetGlobal and SetGlobal callbacks, an XCMD can get or set the value of any variable defined in the context of the call, including local variables. Using EvalExpr, SendCardMessage, or SendHCMessage, an XCMD can execute any valid UserTalk script.

XCMD glue scripts

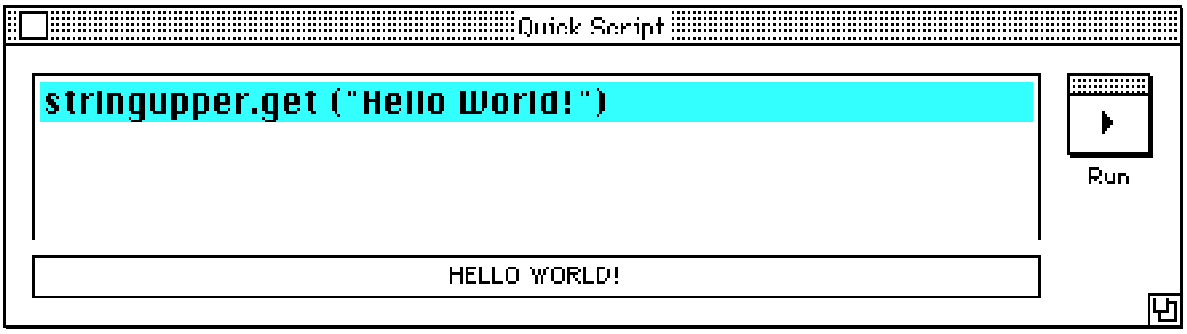
Glue scripts for XCMDs work much the same way as glue scripts for UCMDs with two exceptions:

1. **Usually there is only one glue script for each XCMD.**
2. **XCMD glue scripts use the callXCMD built-in.**

Here's what the stringupper XCMD glue script looks like:



You can call it from the Quick Script window as follows:



See "UCMD glue scripts," above, for more information. For background on glue scripts, see the "Frontier Install File Creator" sub-folder in the Utilities & Scripts folder.

Here’s the complete list of supported XCMD callbacks:

- EvalExpr
- SendCardMessage
- SendHCMessage
- SendHCEvent
- GetGlobal
- SetGlobal
- ZeroBytes
- ScanToReturn
- ScanToZero
- StringEqual
- StringLength
- StringMatch
- ZeroTermHandle
- BoolToStr
- ExtToStr
- LongToStr
- NumToHex
- NumToStr
- PasToZero
- PointToStr
- RectToStr
- ReturnToPas
- StrToBool
- StrToExt
- StrToLong
- StrToNum
- StrToPoint
- StrToRect
- ZeroToPas

Sample UCMDs & XCMDs

stringOps UCMD

This UCMD implements a single Apple Event, it converts the single string parameter to upper case.

Enter the following into the Quick Script window. It displays HELLO WORLD! in the main window.

```
msg (stringOps.allUpper ("Hello World!"))
```

scriptRunner UCMD

Demonstrates the use of the runscript routine in the UCMD Toolkit.

The scriptRunner.do glue script takes one parameter, a string containing a UserTalk script which will be executed.

Enter the following into the Quick Script window. It displays a random number between 1 and 100 in Frontier's main window.

```
scriptRunner.do ("msg (random (1, 100))")
```

trigCmd UCMD

This is the UCMD version of the Trig sample application.

Check out trigCmd.examples for performance testing script and a oneLiners outline.

wordInfo UCMD

This UCMD implements three Apple Events that locate and count words in a string of text. Check out wordInfo.examples for a oneLiners outline and a script that tests UCMDs for memory leakage.

Note: we've also included wordInfo as an XCMD so you can easily compare the two different approaches.

stringupper XCMD

This XCMD converts the single string parameter to uppercase.

Enter the following into the Quick Script window. It displays HELLO WORLD! in the main window.

```
msg (stringupper.get ("Hello World!"))
```

This XCMD duplicates the stringOps UCMD. We used it in determining the performance differences between XCMDs and UCMDs.

syslargestblock XCMD

This XCMD returns the size of the largest block in the Macintosh system heap. It's the same number that's displayed by the About This Macintosh command in the Finder's menu.

To call this XCMD from a script, we've provided a glue script that's called as follows:

```
syslargestblock.get ()
```

sysfreemem XCMD

This XCMD returns the total number of free bytes in the Macintosh system heap.

To call this XCMD from a script, we've provided a glue script that's called as follows:

```
sysfreemem.get ()
```