DTS QUICKTIME UTILITIES (QTU)

Version and Version Numbering
4/2/95

The folder title reflects the date when the kit it released, for example “QTU March-95”.

Introduction
This is a collection of QuickTime related functions. In some cases you could reuse them as they are. In some cases the example is rather a cookbook example showing one technique of many, so eventually you need to modify the code for your own use. This kit also contains a couple of smaller frameworks that could be used for quick testing of QuickTime utilities.

Legal Statements
Copyright © 1994-1995 Apple Computer, Inc. All rights reserved.

You may incorporate this sample code into your applications without restriction, though the sample code has been provided "AS IS" and the responsibility for its operation is 100% yours. However, what you are not permitted to do is to redistribute the source as "DSC Sample Code" after having made changes. If you're going to redistribute the source, we require that you make it clear in the source that the code was descended from Apple Sample Code, but that you've made changes.

Error Reporting
If you encounter problems with this code, send a report stating:

- Name of the sample (and the source code file)
- How the error is reproduced
- What source code lines in the sample are problematic
- What version of the sample you have, or from what source did you obtain the sample
- Your environment, specifying:
    - System environment (System 7.0, 7.1, 7.5 and so on)
    - QuickTime environment (QT 1.6.1, 2.0 and so on)
    - Any additional extensions in your system
    - Hardware environment
    - Development environment (including version level)
- Any other valuable feedback and comments

Send this report to sandvik@apple.com.

Build Environment
The code has been built and tested using the following environments:

- Universal Headers 2.0a3 (ETO #16) with MW 5.5 enhancements
- QuickTimeLib PEF file (ETO#16, MW 5.0 distribution) when compiling native code
- Metrowerks 5.5 GM, PPC and 68k environments

If you want to use Think or MPW, move the DTSQTUtilities files to the project file, and create makefiles or project files that include the right files and libraries. Lack of time for testing made it unable to ship such project and make files at this point of time.

How to Build and Test

Select the Metrowerks project file if you want to rebuild the samples.

Run various applications included in this folder that tests the functions specified in the DTSUtilities files. In some cases maybe a particular configuration for a framework is missing (lack or time or mistake), or causes known or unknown problem. If so, sorry, and we want to know about this.

If you want to make a fat binary, then copy the CODE, DATA and XREF resources from the 68k binary to the PPC one. Note that you need to specify weak linking when you compile Metrowerks projects using the QuickTimeLib PEF file. This is done by selecting the funny triangle next to the QuickTimeLib entry in the project (this is default with the shipped projects).

Movie Framework

One of the bigger examples is a movie framework that is just a simple application environment for playing back movies. Many of the examples are used directly in either this framework, or then the framework is modified in various settings. If a function or class is generic or shows a generic technique, then it's included in the main library. If the application needs generic toolbox support, then this is not included.

For example the library has a generic function showing how to print movie posters, however event handling and MoviesTask is handled inside the actual framework. The idea is to separate the QuickTime related functionality from a basic framework that has already been written N times by DTS and various developers.

In other words in most cases you could use your own framework or a framework provided by various development tools (PowerPlant, MacApp, TCL and so on).

Drag and Drop Framework

This is a simple Applevent driven frame work that understands the open, print, open new and quit AEs. This framework could be used for quickly writing simple droplet applications, you could drag a movie or many movies and do something with each one of them. Or control this using an Applescript sequence.

Resources

The libraries use no whatsoever resources. It is up to the framework or application environment to specify Alerts, window definitions and so on. The reasoning is that generic code should not rely on external resources if possible.

Error Handling

Most functions will return a specific OSErr (error number) any of the toolbox calls encounter errors, otherwise a boolean true or false is return. If the *debug* flag is enabled then any errors returned from the toolbox will ensure that the errors are presented immediately. In the case of Macintosh we will display the error in a MacsBug message (DebugStr).

It is up to the user of the functions to handle error cases, either by terminating or making a choice how to continue after a function has not successfully completed (if a noErr is *not* returned). You could add more error checking in the library code if you want to make sure that most or all error conditions are immediately handled.

The code uses a macro called DebugAssert. If the *debug* flag is enabled then this code is compiled. DebugAssert will test a statement, and if this statement is false then it will trigger a debugging window showing the file and line where the statement is false.

#Defines
The code has the following defines:

powerc
This signals that the following code will be compiled for PPC use.

DEBUG
This indicates that the code block is enabled for debug builds only.

macintosh
This defines that the code block is for Macintosh compilers and environment.

USESIOUX
This is a specific Metrowerks related flag that indicates that the sioux standard i/o package is used for creating a text based output window. Deep inside MacFramework.c this enables us to pass events to the sioux functions for scrolling and similar things, and MacMain has a section for disabling or enabling specific SIOUX window properties as well.

__MWERKS__
This is the flag that the Metrowerks environment will enable if you compile code with CodeWarrior. It is used in cases to include files that are specific to CodeWarrior (for example SIOUX).

Organization of the Functions
The DTSQTUtilities.h file has the basic constants, macros and function prototypes defined. The DTSQTUtilities.c file has the actual code. This file also has the full description of each function:

/ Name of the Function - Pattern definition, in other words what does this function show.*

Function Prototype

Values passed to the function

Description of what the function does, and other things good to know.

Example of use.
** /*

Example:

```
/*  QTUDoIgnoreDrags - Disable Drag and Drop facilities of the movie
    controller environment.

pascal OSErr  QTUDoIgnoreDrags(MovieController  mc)

mc                Is the specified moviecontroller to be used.

DESCRIPTION
```

```
        QTUDoIgnoreDrags will ensure that the Drag and Drop functionality is not
        handled within the movie specified by the movie controller.

ISSUES
        Note that drag and drop with movie controllers are supported starting
        with QT 2.0.
*/
```

All functions start with the prefix *QTU* (*Q*uick*T*ime *U*tilities). This way by reading the sample code examples it's easy to see where the function is defined, and we avoid a little bit of the name space problems if someone takes the code as it is and uses it in other projects.

If possible the code is using similar naming conventions as QuickTime code in general, examples of such variable names are mc (movie controller), theMovie (specific movie used), tr (time record) and so on).

How to use the MovieShell Framework

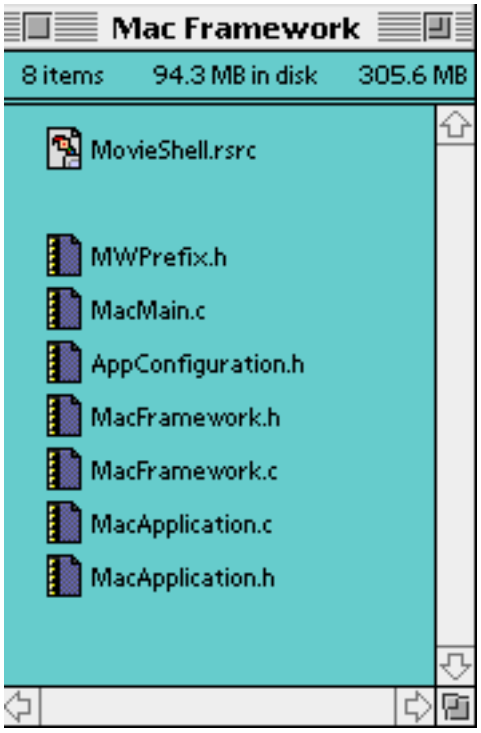
1. Copy the folder and rename it for the project.

2. Rebuild the sources, make sure that possible source file paths and similar are properly resolved. Note that the project depends on the DTSQTUtilities .c and .h files, so you need to either reinclude these in your project environment, or then copy these into the same folder.

3. Most of the application code is hidden inside the Mac Framework:

You need to go in and change some of these files, but let's start from the top. The TestFunctions.c and .h files is the main file combination for quickly adding test functions to the project. Note that we have two different project files one for the Metrowerks PPC environment and the other for the 68k environment.

In most cases you add test code to this TestFunctions.c/TestFunctions.h file, or then create more files and add these to the project.



The Mac Framework hides most of the basic code for displaying movies and using the movie controllers. In most cases you don't need to touch the MacFramework files. However you most likely need to work with the other files.

Use the *MacApplication* to override issues related to the basic behavior. The most common one is to go in and define what happens when a menu is selected, as in:

```
void HandleApplicationMenu(short theMenuID, short theMenuItem)
{
	//  @@@ HANDLE ANY ADDITIONAL MENU ENTRIES HERE
	switch(theMenuID)
	{
		case mTesting:
			switch(theMenuItem)
			{
				case iTest1:
				{
				// @@@ INSERT YOUR TEST FUNCTION 1 HERE
					ResizeTheMovieWindow(kNormalMovieSize);
					break;
				}
```

This file also has other functions and globals you might want to override concerning how to handle idle time, creation of windows, drawing and updating of windows, updating of menus, handling of content click and so on.
The menu enabling and disabling is done inside the MacApplication.c file. The resource file -- MovieShell.rsrc -- has a default additional menu, Testing, and it contains three test entries. Inside the MacApplication code the menu is mTesting, and the menu entries are labeled as iTest1, iTest2... up to iTest10. You could modify these labels if you want to.

MacMain is the main entry point and has the main() function defined.

AppConfiguration.h file has constants that could globally change the behavior of the framework.

MWPrefix is a Metrowerks specific file that could be used to define both global defines as well as indicate what precompiled headers we want to use.

4. So in most cases you could go and change the functions triggered from the menu as well as enabling/disabling of the menus, look at the existing code that changes the size of the movie displayed. After this you write additional test applications that do something based on the movie or some other variable. To setup this environment should not take more than 2 minutes, and this was the reasoning behind this framework, quick and dirty testing of movie toolbox and movie controller related issues.

Credits

The frameworks have code and ideas created by excellent DTS engineers, John Wang, Nick Thompson, Larry Lai, and various other DTS engineers that have worked or work with QuickTime. Peter Hoddie, Jim Batson, David Van Brink and various other excellent QuickTime engineers have also provided code ideas and concepts that are either recycled or reused. Various other QuickTime engineers, past and present, have their fingers in this code. In some cases existing QuickTime samples from the SDK CDs have been reused or factored so that common code have been included into the DTSQTUtilities library. Finally we have had of course developer feedback that have caused us to write code to test out things, and this feedback has found its way to this library.

Current guardian of this pile of stuff: Kent Sandvik/DTS, sandvik@apple.com.