Adding QuickTime MP support to a component.

QuickTime MP works by capturing components registered at extension time and acting as a filter to calls subsequently passed by QuickTime to the component.

If a multiprocessing system is present, frames or parts of frames will be automatically distributed by QuickTime MP to the other processors in the system leaving the main processor free to fetch data, initiate more component calls via QuickTime, or whatever else needs to be done. If all the other processors are busy QuickTime MP processes the frame on the main processor.

QuickTime MP requires components to support two specific component selectors. The kComponentTargetSelect selector and the kComponentGetMPWorkFunctionSelect selector.

Responding to the kComponentTargetSelect selector involves remembering the ComponentInstance of the component doing the targetting -in this case QuickTime MP. This component will be called later in order to retrieve a 'work function' that should be used to perform the actual component work.

Responding to the kComponentGetMPWorkFunctionSelect selector involves returning a pointer to the 'work function' that the component normally uses to do its work and an arbitrary reference constant to be passed to the work function when it is called.

The work function is what the component uses to perform the bulk of its work. For example the BandCompress function is frequently architected such that it establishes what work needs to be done and then either directly calls a 'CompressStrip' function, or defers the call using the deferred task manager. The 'CompressStrip' function or its equivalent is the 'work function'.

Rather than access the function by name when it is needed, a pointer to the work function should be acquired by calling the new component manager function ComponentGetMPWorkFunction(). Components that are 68K based or never received the kComponentTargetSelect selector should simply specify themselves as the Component Instance from which the work function is to be retrieved. Otherwise, if the component is PPC, and the component was targetted, it should specify the targetting component as the Component Instance from which the work function is to be retrieved. The pointer thus acquired should be used to perform the work.

When your component calls the work function retrieved using the above technique it really calls QuickTime MP. QuickTime MP directs an MP task to call the real work function using the parameters you specify and the refcon you provided in response to the kComponentGetMPWorkFunctionSelect selector. Note that because the work function is going to be called from an MP task it must not contain ANY toolbox calls and MUST be PowerPC based.

A work function has a well-defined entry point:

```
typedef pascal ComponentResult (*ComponentMPWorkFunctionProcPtr)
```

```
(void *globalRefCon, ComponentMPWorkFunctionHeaderRecordPtr header);
```

`globalRefCon` is the refcon value specified in your response to the kComponentGetMPWorkFunctionSelect selector. It will probably be a pointer to your global storage handle.

`ComponentMPWorkFunctionHeaderRecordPtr` is a structure set up immediately prior to calling the work function. It is used for passing certain required information as well as the normal parameters needed by your work function (e.g. baseAddr, rowBytes, etc).

The required information is as follows:

```
struct ComponentMPWorkFunctionHeaderRecord {
      UInt32      headerSize;
      UInt32      recordSize;
      UInt32      workFlags;
      UInt16      processorCount;
      UInt16      unused;
};
```

`headerSize` should be set to:
`sizeof( ComponentMPWorkFunctionHeaderRecord )`

`recordSize` should be set to:
`header.header.headerSize + sizeof( WorkParameters );`
where `WorkParameters` is a user defined structure that can hold the arbitary parameters to be used by the work function.

`workFlags` should be set to one of:
`mpWorkFlagDoWork`
`mpWorkFlagDoCompletion`
`mpWorkFlagCopyWorkBlock`
`mpWorkFlagDontBlock`
`mpWorkFlagGetProcessorCount`

`mpWorkFlagDoWork` means that the work function is is to be called or is being called for the purpose of doing work and that the user defined parameters describe the work to be done. Your work function should respond to the presence of this flag by performing the requested work.

`mpWorkFlagDoCompletion` means that the work function is to be called or is being called for the purpose of 'completing' after a frame has been finished. In the case of compression and decompression this will mean calling `CallICMCompletionProc()` using the completionProc which should be recorded and obtained from the user defined `WorkParameters` section. Note that if a component has NOT been targetted then it should call the work function itself with this flag set. If it has been targetted then QuickTime MP will automatically call the work function with this flag set when the frame is finished. Note that the latter will occur at interrupt time.

`mpWorkFlagCopyWorkBlock` should be set prior to calling the work function if the `ComponentMPWorkFunctionHeaderRecord` will possibly be destroyed before the work function runs, i.e. it is stored locally to the function that is calling the work function. Since MP tasks run asynchronously the record must remain valid at all times. If the record is allocated out of global memory and is not released until the completion proc is called then you do not need to set this flag.

`mpWorkFlagDontBlock` should be set if QuickTime MP should not wait for preceeding frames in the sequence being grabbed or played to complete. This is important if the following flag is to be used.

Problem: differential frames in jeopardy in scenario described below.

`mpWorkFlagGetProcessorCount` can be used to get the number of processors in the system. QuickTime MP will return the number in the `processorCount` field of the `ComponentMPWorkFunctionHeaderRecord`. This number can be used to submit n seperate work requests for a single frame where n is the number of processors available. This should increase the playback rate of a single frame. The `mpWorkFlagDontBlock` should be set when submitting work requests for split frame data.

Problem: differential frames may be generated out of sequence since no blocking will occur on previous frames.

To incorporate the work function specific parameters into a single block to pass to the work function you can use the following approach:

```
/* Your work function parameters */
typedef struct    {
      char *data;
      char *baseAddr;
      short rowBytes;
      ICMCompletionProcRecord completionProc;
      .
      etc...
      .
} WorkParameters, *WorkParametersPtr;

/* Required data format for a work function */
typedef struct    {
      ComponentMPWorkFunctionHeaderRecord header;
      WorkParameters params;
} WorkHeader, *WorkHeaderPtr;
```

A work header pointer is what is passed to the work function.

The example code shows all of the changes described in this document. It is based on the codec sample provided on the QuickTime 2.1 CD. The CompressStrip and DecompressStrip functions were altered so that they could do a whole frame at once -the more work that you can do in one call to the work function the better.

All of the significant alterations are denoted with the comment:

```
/* --- Add for •QuickTime MP */
```

Additional comments within the code may help clarify things.

For early adopter support please call:

Chris Cooksey
(770) 967 2077 x213
ccooksey@daystar.com