QuickTime 2.2 (?)

Miscellaneous Changes

Notes on new stuff in QuickTime 2.2. Some of the information may even be accurate. Some of the features may not actually be in QuickTime 2.2.

Image Compression Manager

ExtractAndCombineImageFields

Many hardware compression formats contain two fields of video data. Examples of this include Radius VideoVision, TrueVision Targa 2000, among others. Many video processing programs need to perform operations on the individual fields, such as reversing them, or combining one field of a frame with a field from anothe frame. These operations have required decompressing each frame, copying the appropriate fields, and then recompressing. In addition to being a slow operation, this also caused a quality loss because the decompress/compress phase. The new ExtractAndCombineImageFields call allows an application to request that the field operation be performed directly on the compressed data. The routine takes as inputs one or two compressed images, and creates a compressed image on output. A long word of flags indicates the operation to be performed

```
pascal OSErr ExtractAndCombineImageFields(long fieldFlags, void
      *data1, long dataSize1, ImageDescriptionHandle desc1, void
      *data2, long dataSize2, ImageDescriptionHandle desc2, void
      *outputData, long *outDataSize, ImageDescriptionHandle
      descOut);
```

The field flags field contains flags indicating the operation to be performed. The flags currently define dare shown below. A correctly formed request will specify one odd and one even input field, and map them one to the even output field and the other to the odd output field.

```
enum {
      evenField1ToEvenFieldOut = 1<<0,
      evenField1ToOddFieldOut = 1<<1,
      oddField1ToEvenFieldOut = 1<<2,
      oddField1ToOddFieldOut = 1<<3,
      evenField2ToEvenFieldOut = 1<<4,
      evenField2ToOddFieldOut = 1<<5,
      oddField2ToEvenFieldOut = 1<<6,
      oddField2ToOddFieldOut = 1<<7
};
```

The “data1” and “data2” fields should point to compressed data of the sizes indicated by “dataSize1” and “dataSize2” respectively. The “desc1” and “desc2” fields contain image descriptions describing the type of compressed data in each field. If The field flags only uses a specifies data to be used from data1, then data2, dataSize2, and desc2 can be nil. The outputData pointer points to a memory buffer to receive the compressed resulting frame. Use GetMaxCompressionSize to determine how big this buffer should be. On input the

“outDataSize” field indicates the size of the outputData buffer. When the call returns, the “outDataSize” field will be updated to contained the actual size of the compressed data. The “descOut” field indicates what format the compressed data should be in. Typically this will be the same format as “desc1” and “desc2”. If no codec is present in the system that can perform the operation, ExtractAndCombineImageFields returns an error of codecUnimpErr. This call is currently implemented for the Component Video (YUV) codec. This should be helpful for debugging.

ExtractAndCombineImageFields for Codec Authors
The real functionality of ExtractAndCombineImageFields is performed by individual Image Codecs. A new codec component routine has been defined for this purpose. Its component selector is shown below.

```
codecExtractAndCombineFields        = 0x15

pascal ComponentResult CDCodecExtractAndCombineFields(PREAMBLE,
      long fieldFlags, void *data1, long dataSize1,
      ImageDescriptionHandle desc1, void *data2, long dataSize2,
      ImageDescriptionHandle desc2, void *outputData, long
      *outDataSize, ImageDescriptionHandle descOut)
```

The parameters for this routine are exactly as explained above for ExtractAndCombineImageFields. The codec should check to make sure that it understands the image formats specified by “desc1” and “desc2”, as these may not be the same as the codec’s native image format. The image format specified by “descOut” will be the same as the image codec’s native image format.

Alpha Channel Support
The Image Compression Manager now understands how to blit images that contain an alpha channel. Alpha channels are only supported for images with a 32 bit source pixel. Alpha channels in 16 bit source pixels are not supported. The high byte of each pixel contains the alpha channel. The alpha channel can be interpreted in one of three ways: straight alpha, pre-multiplied with white, and pre-multiplied with black. Alpha channels are supported by graphics mode (transfer mode). QuickDraw graphics modes are only 8 bits in size. QuickTime graphics modes start at 256. The graphics modes for alpha channel blitting are shown below.

```
enum {
      graphicsModeStraightAlpha = 256,
      graphicsModePreWhiteAlpha = 257,
      graphicsModePreBlackAlpha = 258
};
```

Currently only the ‘raw ‘ and animation codecs support storing data with an alpha channel, both in the “Millions of Colors+” mode.

To set a track in a movie to use an alpha channel graphics mode use MediaSetGraphicsMode. To set an Image Sequence to use an alpha channel graphics mode (or any other) use SetDSequenceTransferMode.

Known issues: Using frame differenced Animation data with an alpha channel may cause interesting results with the current code. Alpha channel support requires the QuickTime PowerPlug.

JPEG & RAW Codecs

Both the JPEG and Raw codecs have been updated to include asynchronous decompress support. This support is only present on the PowerPlug versions, as the performance on 68k machines would be less than exciting.

Sprites

Both the Sprite Toolbox and the Sprite Track now support graphics modes. This allows for sprites to blend with the background in various ways. Of particular interest is the ability to use 32 bit animation compressed sprites with an alpha channel graphics modes.

The graphics mode is specified to both the Sprite Toolbox and the Sprite Track using ModifierTrackGraphicsModeRecord which contains both a graphics mode and an opcolor (which is required for some graphics modes, such as blend). To set the graphcis mode when using the Sprite Toolbox, use the following constant when calling SetSpriteProperty and pass a pointer to a ModifierTrackGraphicsModeRecord structure.

```
kSpritePropertyGraphicsMode
```

To include the graphics mode in a Sprite Track, include an atom of type kSPritePropertyGraphicsMode (and data of type ModifierTrackGraphicsModeRecord).

The sprite track also supports receiving modifier track data to control the sprites. Currently only matrix inputs are supported. These can be sent to individual sprites to control their location. If the movie also contains matrices to move the sprites, the results are undefined (i.e. don't try this). To do this, set up a modifier track (such as a Tween Track) to send matrix data to the sprite track. In the input map use the following input type to indicate that the matrix should be set to a sprite.

```
#define kTrackModifierObjectMatrix 6
```

In the input map, the "kSpritePropertyImageIndex" contains a long indicating the id of the sprite to send the data to. For example, to send the data to sprite id 3, set the kSpritePropertyImageIndex to 3. (I know the name is confusing, but is the right idea).

Movie Toolbox

One nifty new Movie Toolbox routine has been added to make it easier to work with modifier tracks. GetTrackDisplayMatrix returns a matrix which is the concatentation of all matrices currently effecting the tracks location/scaling/etc. This includes the movie's matrix, the track's matrix, the modifier matrix, and any other random matrix that might be around (none at this time). Since modifier information is passed between tracks at MoviesTask time, the information returned by this call will only be as accurate as the time of the last MoviesTask call.

```
pascal OSErr GetTrackDisplayMatrix( Track theTrack, MatrixRecord
      *matrix )
```

It is already possible to determine the entire clip of a track at the current time using GetTrackDisplayBoundsRgn. The results of GetTrackDisplayBoundsRgn take into account any clip regions provided by modifier tracks.

Data Handler

QuickTime's primary data handler (the HFS Data Handler) has been updated to allow for higher performance play back. Specifically, in the past reads were only started from MoviesTask. Those reads were asynchronous, but there could be considerable gaps between the completion of one read and the start of the next. The data handler has been modified to allow a read to be started from the completion of the previous read. This allows the data handler to maximize the throughput from the device. Noticeable performance improvements have been observed.

Sequence Grabber

The sequence grabber sound channel has been enhanced to allow sound to be capture at any sample rate. The sample rate is specified, as in the past, by using SGSetSoundInputRate. However, if the requested rate is not close to one of the hardware rates, the sound will be software rate converted to the requested rate. This feature is important to QuickTime Conferencing. The user interface in the sequence grabber sound sample panel has been updated for this feature, but does not fully expose it. If 8k is not present in the sound input driver's native rates, 8k is added to the rate pop-up. The user is not current allowed to select any arbitrary rate, just harware rates and 8k.

QT Resource Manager

The QuickTime resource manager is designed to address the needs of QTi to be able to store data in a variety of location, while accessing it in the same way regardless of where it is stored. The resource manager groups data into resource maps. Each map contains a list of resources, and information about where the actual resource data is stored. The data may be stored in the map itself, or the data may be stored in the same file the map is stored in (but external to the map itself). Finally, the data may be stored external to the map by placing it in an external data reference. Another feature of the QuickTime resource manager is that resource maps may be chained together into a tree. If a resource cannot be found in a given map, its parent map will be searched for the data, and so on, until the root is reached.

Resources are identified by four long words. The first is class. This is usually the object class of the object that owns the resource. The next two longs are type and subtype. The last long is the resource id within the class/type/subtype. Another feature of the QuickTime resource manager is that if a search for a resource fails, the class field is replaced with the parent class and the search is repeated. In this way, for example, if a resource cannot be found for a picture object, the search will be retried on the base object. This allows resources to be overriden in the same way object methods are typically overriden.

Because searching for a resource can be a relatively expensive operation, the QuickTime resource manager requires the application to create a reference to a resource in a particular map. In this way the resource can be accessed many times, but only looked up once. QuickTime ensures that the reference is invalidated as appropriate due to changes in the resource maps, so it will always be accurate. This allows an object, for example the picture object, to create a reference to its required resources once when being initialized and then quickly access them for use when necessary.

Because the QuickTime resource manager has information about all resources currently in use, and when they were last used, it can perform reasonably intelligent caching of resource data.

Note: the current implementation of the QuickTime resource manager is not as clever, fast, or wonderful as it might be. It will be improved once its usage can be reasonably profiled.

```
pascal OSErr QTNewResourceMap(ResourceMap *theMap, ResourceMap
      parentMap)
```

QTNewResourceMap creates a new resource map. A parentMap may be specified. If there is no parent map, the parentMap field should be set to nil.

```
pascal OSErr QTDisposeResourceMap(ResourceMap theMap)
```

QTDisposeResourceMap disposes of the specified resource map, all of its resources, all references to resources in the map, and all the map's children maps.

```
pascal OSErr QTRemoveAllResourcesFromMap(ResourceMap theMap)
```

QTRemoveAllResourcesFromMap empties all resources out of the specified map. This is useful if you need to reinitialize an existing map. All references to resources in the map remain.

```
pascal OSErr QTAddDataReferenceToMap(ResourceMap theMap, OSType
      dataRefType, Handle dataRef, ResourceDataReference
      *resDataRef)
```

Before adding a resource to the map that stores its data external to the map using a data reference, it is necessary to add the data reference to the map. QTAddDataReferenceToMap takes a resource map, a data reference (specified by dataRefType and dataRef), and returns a ResourceDataReference. After all resources have been added to the map that require the data reference, the ResourceDataReference should be disposed using QTRemoveDataReferenceFromMap. ResourceDataReferences are maintained in the resource map using counters. Calling QTAddDataReferenceToMap adds the data reference, and sets it counter to 1. As resources are added that use the data reference, its counter is incremented. When the application done adding the resources to the map that use the ResourceDataReference, calling QTRemoveDataReferenceFromMap decrements the counter, ensuring that the counter of the number of resources that use the data reference is correct. It is important that this counter be correct so that the data reference can be removed from the map when it is no longer referenced by any resources.

```
pascal OSErr QTRemoveDataReferenceFromMap(ResourceDataReference
      resDataRef)
```

See the description of QTAddDataReferenceToMap.

```
pascal OSErr QTAddResourceReferenceToMap(ResourceMap theMap,
      OSType rClass, OSType rType, OSType rSubType, long id, long
      offset, long size, ResourceDataReference resDataRef)
```

QTAddResourceReferenceToMap creates a resource entry in the resource map. The resource is identified by the rClass, rType, rSubType, and id fields. The resDataRef field is the data reference added to the map by QTAddDataReferenceToMap. The offset and size indicate the offset of the data into the resource map, and the size of the data in the resource map. A size of “-1” indicates that the data is the all the data in the file beginning at “offset”. This is a convenient way to create a resource which references data which takes up an entire file, and whose size is may change after the movie is authored. QuickTime is more efficient at run time if the actual size of the resource is specified instead of -1.

```
pascal OSErr QTAddResourceDataToMap(ResourceMap theMap, OSType
      rClass, OSType rType, OSType rSubType, long id, void *data,
      long size)
```

QTAddResourceDataToMap adds data directly into a resource map instead of it being stored external to the map. This maps the map larger, and precludes unloading the resource data under low memory conditions, so it should be used carefully. The resource is indentified by rClass, rType, rSubType, and id. The resource data is pointed to by the “data” field and the resource size is indicated by “size”.

```
pascal OSErr QTAddResourceAliasToMap(ResourceMap theMap, OSType
      rClass, OSType rType, OSType rSubType, long id, OSType
      rToClass, OSType rToType, OSType rToSubType, long toID)
```

Current unused. Intended to be a mechansim to allow a given resource to specify that its data actually resides in another resource.

```
pascal OSErr QTRemoveResourceFromMap(ResourceMap theMap, OSType
      rClass, OSType rType, OSType rSubType, long id)
```

QTResourceFromMap deletes the specified resource from the map. Any resource references which reference this resource will be invalidated.

```
pascal OSErr QTNewResourceReference(ResourceMap theMap, OSType
      rClass, OSType rType, OSType rSubType, long id,
      ResourceReference *resourceRef, QTiObject notifyOnChanges)
```

QTNewResourceReference creates a reference to the resource specified by rClass, rType, rSybType, and id relative to “theMap”. The resource reference is returned in “resourceRef”. The “notifyOnChanges” field is currently unused.

```
pascal OSErr QTNewResourceReferenceForObject(QTiObject theObject,
      OSType rType, OSType rSubType, long id, ResourceReference
      *resourceRef)
```

QTNewResourceReferenceForObject is similar to QTNewResourceReference, but is intended for use by authors of QTi objects. The rClass, theMap, and notifyOnChanges fields are replaced by “theObject” which indicates which object is creating the reference. From the object, QuickTime determines the values for the missing fields.

```
pascal OSErr QTDisposeResourceReference(ResourceReference
      resourceRef)
```

When a resource reference is no longer in use, QTDisposeResourceReference disposes it. Resource references created with either QTNewResourceReference or QTNewResourceReferenceForObject can be passed to this routine.

```
pascal OSErr QTGetResourceSize(ResourceReference resourceRef, long
      *dataSize)
```

QTGetResourceSize returns the size of the resource specified by the resourceRef. If the resource was defined with a size of “-1”, this routine will return the actual resource size, not “-1” based on the rules described under QTAddResourceDataToMap.

```
pascal OSErr QTLockResourceData(ResourceReference resourceRef,
      long offsetIntoResource, long dataSize, Ptr *where)
```

QTLockResourceData provides a way to directly access the data stored in a particular resource. The starting offset of the desired data in the resource is specified, as well as the data size. A pointer is returned to the actual resource data. This routine can fail, due to insufficient memory. In these cases, the caller cal use QTReadResourceData (described below) to access the resource data. UseQTUnlockResourceData to release the data. It is important to unlock the resource data as soon as possible to avoid heap fragmentation. It is safe to call QTLockResourceData for multiple segments of resource data, as long as each is eventually balanced with a call to QTUnlockResourceData.

```
pascal OSErr QTUnlockResourceData(ResourceReference resourceRef,
      long offsetIntoResource, long dataSize)
```

QTUnlockResourceData releases the resource data previously locked down. Caching is performed by the QuickTime resource manager so that if the data is asked for again later, it will most likely still be in memory. The same values for offsetIntoResource and dataSize should be used as when QTLockResourceData was called.

```
pascal OSErr QTReadResourceData(ResourceReference resourceRef,
      long offsetIntoResource, long dataSize, Ptr where)
```

QTReadResourceData copies a segment of resource data into the callers buffer. The offset into the resource data and size of the resource data is provided. A pointer to a buffer is also

passed. QTReadResourceData is much less likely to fail than QTLockResourceData, as the QuickTime resource manager is not required to allocated any memory.

```
pascal OSErr QTGetResourceDataReference(ResourceReference
      resourceRef, OSType *dataRefType, Handle dataReference)
```

QTGetResourceDataReference returns the data reference associated with the given resource reference. If there is no data reference, because the data is stored in the map itself, the dataRefType field is set to nil. Most applications will not need to use this routine.

```
pascal OSErr QTPutResourceMapIntoAtom(ResourceMap resourceMap,
      QTAtomContainer container, QTAtom atom, OSType dataRefType,
      Handle dataRef)
```

QTPutResourceMapIntoAtom flattens the specified resourceMap into a QTAtomContainer at the specified QTAtom node. The resource map is stored in a form suitable for storage. The resource map can be recreated from the QTAtomContainer/QTAtom node by using either QTNewReourceMapFromAtom or QTMergeAtomToResourceMap. If the file that the resource map is to be stored in is know, its data reference should be passed into the dataRefType and dataRef fields. This allows the QuickTime resource manager to create relative aliases to the data, so that data references can be resolved more accurately.

```
pascal OSErr QTNewResourceMapFromAtom(ResourceMap *resourceMap,
      ResourceMap parentMap, QTAtomContainer container, QTAtom
      atom)
```

QTNewResourceMapFromAtom creates a new resource map from the specified flat resource map stored in the QTAtomContainer within the QTAtom node. In most cases, QTMergeAtomToResourceMap should be used.

```
pascal OSErr QTMergeAtomToResourceMap(ResourceMap resourceMap,
      QTAtomContainer container, QTAtom atom, OSType dataRefType,
      Handle dataRef)
```

QTMergeAtomToResourceMap takes a resource map, and merges the contents of the flat resource map contained in QTAtomContainer within QTAtom node. If possible, the data reference of the file that the resource map was loaded from should be specified in dataRefType and dataRef to ensure that external data references can be resolved.