Technical Note TN1195
Tagging Handle Data References in QuickTime 4

CONTENTS

This Technote discusses the use of data reference extensions, which are used by QuickTime 4 to tag handle data references. Data reference extensions give you the ability to associate Mac OS file type, MIME type, initialization data and/or file name information with a given file.

This Note is directed at developers who want to take advantage of this new feature in QuickTime 4.

Updated: [Apr 03 2000]

Overview

With QuickTime 4, it is now possible to tag a handle data reference in such a way that the data reference is associated with an equivalent Mac OS file type, MIME type, and/or file name (needed for the extension). You can also specify initialization data for the handle data reference.

After tagging the data reference, any QuickTime function call that will look at the type of data (`GetGraphicsImporterForDataRef`, `NewMovieFromDataRef`, `GetMovieImporterForDataRef`, etc.) will find the type information you provided.

For example, you can tag a handle to a PhotoShop image file's data with the Mac OS `'8BPS'` file type and be assured that the correct Graphics Importer Component will be chosen.

The process of tagging a handle data reference involves adding what we call data reference extensions, which are new with QuickTime 4. Currently, extensions are defined to allow the tagging of the Mac OS file type, MIME type, and/or file name (actually, QuickTime 3 recognized any file name extension added to the data reference).

Also, the handle data handler is the only data handler that supports the above extensions since there's no intrinsic file type, file name or MIME type for a block of memory.

Back to top

Data References

The handle data handler is the only data handler that supports the data reference extensions. So what exactly is a handle data reference? A handle data reference is simply a reference to data that resides in a handle in memory, rather than in a file (or somewhere else).

In QuickTime, all data handlers identify their data with data references. Data references specify the location of the data, and the data reference type indicates the data handler able to interpret the data. We can therefore specify a data reference programatically by a type value and a reference value as follows:

Table 1 - Specifying a Data Reference

OSType dataRefType	Specifies the type of data reference
Handle dataRef	Specifies the actual data reference

The `dataRef` value specifies the actual data reference. This is a handle to the information that identifies the data to be used. The type of information stored in the handle depends on the data reference type. For example, if your application is loading a movie from a file, this handle would contain a Macintosh alias to the movie file.

The `dataRefType` value specifies the type of data reference. For example, for an alias data reference you set this to `rAliasType ('alis')` , indicating that the reference is an alias.

Listed below are all the available data reference types:

Table 2 - Data Reference Types

Data reference type	Description
`'alis'`	Data reference is a Macintosh alias handle. An alias handle contains information about the file, including its full path name. For more information about aliases, see Inside Macintosh: Files.
`'hndl'`	Data reference is a Macintosh handle whose data is the handle containing the data. For more information about Macintosh handles, see Inside Macintosh: Memory. In addition to specifying movie data in memory, this handle may contain data reference extensions as described in section "Data Reference Extensions" below.
`'rsrc'`	Data reference is a Macintosh alias handle. However, appended to the end of the alias handle is the resource type (stored as a 32-bit unsigned integer) and ID (stored as a 16-bit signed integer) to use within the specified file. Both these values must be big-endian format.
`'url '`	Data reference is a handle whose data is a C-string (null-terminated) specifying a URL (note: a url data reference can have data reference extensions too, so the actual size of a url data reference may be larger than the length of the string).

Back to top

Constructing Data Reference Handles

How do you construct a data reference handle? As stated above, the information that identifies the data is specified in the data reference handle (while the type of the data reference is specified in the data reference type). Here's how to construct data reference handles for the various data references:

Alias Data Reference Handle

The alias handle data reference handle is a standard Macintosh alias handle (see *Inside Macintosh:Files* for more information). Here's code showing how to construct the data reference handle for an alias data reference. In our example, you pass a standard Macintosh file system specification record for the desired file in the `theFile` parameter and an alias handle is returned. We use the `NewAlias` function to create the alias handle, but you can also use any of the other Alias Manager functions which return an alias handle (`QTNewAlias`, `NewAliasMinimal`, `NewAliasMinimalFromFullPath`, etc.):

```
Handle MyCreateAliasDataReferenceHandle(FSSpec *theFile)
{
 OSErr err;
 AliasHandle theAlias;

 err = NewAlias(nil, theFile, &theAlias);
 if (err == noErr)
 {
  return (Handle)theAlias;
 }

 return nil;

}
```

Handle Data Reference Handle

The handle data reference handle is a Macintosh handle whose data is the handle containing the actual data. Here's code showing how to construct the data reference handle for a handle data reference. In our example, you simply pass a handle to your actual data in the `theDataH` parameter and a data reference handle is returned:

```
Handle MyCreateDataReferenceHandle(Handle theDataH)
{
 Handle dataRef = nil;
 OSErr err;

 // create a data reference handle for our data
 err = PtrToHand( &theDataH, &dataRef, sizeof(Handle));

 return dataRef;
}
```

Note:
Note: most QuickTime functions which accept data references accept the data reference handle and data reference type as separate parameters. For example, here's how you would pass a handle data reference to the Graphics Importer function `GetGraphicsImporterForDataRef`:

```
void MyGetGraphicsImporterForDataRef(Handle ourData)
{
 OSErr err;
 ComponentInstance gi;
 Handle dataRefH = nil;


 dataRefH = MyCreateDataReferenceHandle(ourData);
 if (dataRefH)
 {
  err = GetGraphicsImporterForDataRef (
                      dataRefH, /* data reference */
                      HandleDataHandlerSubType, /* data ref type */
                      &gi);

     /* ...now use the returned graphics importer component
      gi as you please... */

     /* close the graphics importer component instance when
  we are done */
  CloseComponent(gi);
    }

    /* clean up our data reference handle when we
       are done */
    if (dataRefH)
    {
     DisposeHandle(dataRefH);
    }
}
```

Note:
Note: you would need to dispose of the data reference handle when you are done.

URL Data Reference Handle

The url data reference handle is a handle whose data is a C-string (null-terminated) specifying a url. Here's code showing how to construct the data reference handle for a url data reference:

```
Handle MyCreateURLDataReferenceHandle()
{
 char    url[] = "rtsp://www.mycompany.com/mymovie.mov";
 Handle  urlDataRef;
 OSErr   err;

 urlDataRef = NewHandleClear(StrLength(url) + 1);
 if ( ( err = MemError()) != noErr) goto bail;

 BlockMoveData(url, *urlDataRef, StrLength(url) + 1);

 return (urlDataRef);

bail:
 DisposeHandle(urlDataRef);
 return nil;
}
```

Resource Data Reference Handle

The resource data reference handle is also a Macintosh alias handle. However, appended to the end of the alias handle is the resource type (stored as a 32-bit unsigned integer) and ID (stored as a 16-bit signed integer) to use within the specified file. These values must be big-endian format.

Here's code showing how to construct the data reference handle for a resource data reference. In our example, we build a data reference for a standard QuickTime movie file whose movie resource resides in the resource fork of the file. Pass a standard Macintosh file system specification record for the desired movie file in the `theFile` parameter. Next, an alias handle is created for this file. Finally, we append to this alias handle the resource type for the movie resource (type `'moov'`) along with the movie resource ID (128). Once this information is appended, the completed resource data reference handle is returned.

Note you can use any of the other Alias Manager functions to create the alias handle (`QTNewAlias`, `NewAliasMinimal`, `NewAliasMinimalFromFullPath`, etc.):

```
Handle MyCreateResourceDataReferenceHandle(FSSpec *theFile)
{
 OSErr err;
 AliasHandle theAlias;

 err = NewAlias(nil, theFile, &theAlias);
 if (err == noErr)
 {
  OSType ourType = EndianU32_NtoB(MovieResourceType);
  short ourID = EndianU16_NtoB(128);

  /* append resource type */
  err = PtrAndHand(&ourType, (Handle)theAlias, sizeof(OSType));

  /* append resource id */
  err = PtrAndHand(&ourID, (Handle)theAlias, sizeof(short));

  return (Handle)theAlias;
 }

 return nil;

}
```

Back to top

Data Reference Extensions

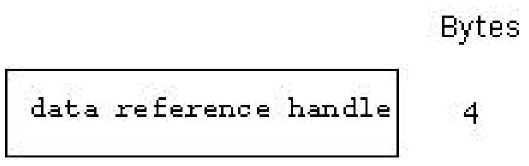
One can certainly construct a handle data reference for a given piece of data, and use this handle data reference with any of the QuickTime API's which operate on data references. However, if you just construct a "plain" handle data reference, the reference doesn't contain any information about the file type or file name, so QuickTime will end up performing a slow validate search to identify the file (since there's really no other way). As well as being slow, this technique will miss some file formats which can't be detected by validation.

For example, when you call the `GetGraphicsImporterForFile` function to try to obtain a graphics importer for a given file, QuickTime first tries to locate a graphics importer component for the specified file based on the Macintosh file type of the file. If it is unable to locate a graphics importer component based on the Macintosh file type, `GetGraphicsImporterForFile` will try to locate a graphics importer component for the specified file based on the file name extension of the file.

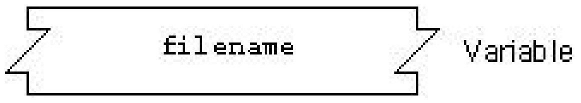
If you have the file name, you should add the file name to the handle data reference. If you know the file type or MIME type, you should add this information as well. You can also specify any initialization data for the handle data reference.

How do you go about adding this information to a handle data reference? Use the handle data reference extensions. Here's the format of a data reference handle with data reference extensions:

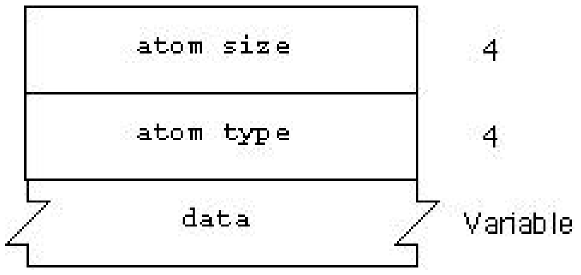
The format of a data reference handle is:



optionally continued by a pascal string containing the file name (*not padded* - zero byte if not available). Note the filename is not really a data reference extension - QuickTime 3 will recognize any file name added to the data reference.



optionally continued by a sequence of n classic atoms, whose format is as follows (atom size and atom type fields must be big-endian):



Atom types may be any of the following:

'ftyp'	Mac OS file type, data is a big-endian OSType
'mime'	MIME type, data is a pascal string (no padding)
'data'	Initialization data, data can be a any block of image, video/audio data, etc.

Note:
If you add any of these atoms, you MUST add a file name first -- even if it is an empty Pascal string.

Back to top

Building a Handle Data Reference with Data Reference Extensions

As shown above, building a handle data reference with data reference extensions involves first creating a data reference handle for your data, then adding any additional data reference extension information (Mac OS file type, MIME type and/or file name, initialization data) that you may have for this data reference handle. Below is a snippet of code which demonstrates how it's done.

You pass to the function first a handle to your data, followed by (optionally) the file name, file type, MIME type and any initialization data you may have. If you specify initialization data, the data is added in-place to the handle data reference (which means the data reference can grow to be quite large). The advantage to adding initialization data in this manner is that once you've created the data reference you no longer have to manage the actual data handle yourself as you would if you added the data via the handle data reference handle.

You may pass nulls for any of the file name, file type, MIME type and initialization data parameters, in which case these parameters are not added as data reference extensions to the data reference handle. The function returns a handle data reference handle for your data:

```
#includeQuickTimeComponents.h
```

```
Handle myCreateHandleDataRef(
     Handle             dataHandle,
     Str255             fileName,
     OSType             fileType,
     StringPtr          mimeTypeString,
     Ptr                initDataPtr,
     Size               initDataByteCount
     )
   {
 OSErr  err;
 Handle dataRef = nil;
 Str31  tempName;
 long  atoms[3];
 StringPtr name;


 // First create a data reference handle for our data
 err = PtrToHand( &dataHandle, &dataRef, sizeof(Handle));
 if (err) goto bail;

 // If this is QuickTime 3 or later, we can add
 // the filename to the data ref to help importer
 // finding process. Find uses the extension.

 name = fileName;
 if (name == nil)
 {
  tempName[0] = 0;
  name = tempName;
 }

 // Only add the file name if we are also adding a
 // file type, MIME type or initialization data
 if ((fileType) || (mimeTypeString) || (initDataPtr))
 {
  err = PtrAndHand(name, dataRef, name[0]+1);
  if (err) goto bail;
 }

 // If this is QuickTime 4, the handle data handler
 // can also be told the filetype and/or
 // MIME type by adding data ref extensions. These
 // help the importer finding process.
 // NOTE: If you add either of these, you MUST add
 // a filename first -- even if it is an empty Pascal
 // string. Under QuickTime 3, any data ref extensions
 // will be ignored.

 // to add file type, you add a classic atom followed
 // by the Mac OS filetype for the kind of file

 if (fileType)
 {
  atoms[0] = EndianU32_NtoB(sizeof(long) * 3);
  atoms[1] = EndianU32_NtoB(kDataRefExtensionMac OSFileType);
  atoms[2] = EndianU32_NtoB(fileType);

  err = PtrAndHand(atoms, dataRef, sizeof(long) * 3);
  if (err) goto bail;
 }


 // to add MIME type information, add a classic atom followed by
 // a Pascal string holding the MIME type

 if (mimeTypeString)
 {
  atoms[0] = EndianU32_NtoB(sizeof(long) * 2 + mimeTypeString[0]+1);
  atoms[1] = EndianU32_NtoB(kDataRefExtensionMIMEType);

  err = PtrAndHand(atoms, dataRef, sizeof(long) * 2);
  if (err) goto bail;

  err = PtrAndHand(mimeTypeString, dataRef, mimeTypeString[0]+1);
  if (err) goto bail;
 }

 // add any initialization data, but only if a dataHandle was
 // not already specified (any initialization data is ignored
 // in this case)
 if((dataHandle == nil) && (initDataPtr))
 {

  atoms[0] = EndianU32_NtoB(sizeof(long) * 2 + initDataByteCount);
  atoms[1] = EndianU32_NtoB(kDataRefExtensionInitializationData);

  err = PtrAndHand(atoms, dataRef, sizeof(long) * 2);
  if (err) goto bail;

  err = PtrAndHand(initDataPtr, dataRef, initDataByteCount);
  if (err) goto bail;
 }

 return dataRef;

bail:
 if (dataRef)
 {
  // make sure and dispose the data reference handle
  // once we are done with it
  DisposeHandle(dataRef);
 }

 return nil;
}
```

Back to top

Additional Notes & Comments

If you add a Mac OS file type or MIME type or initialization data extension as shown above, you MUST add a file name *first* in the sequence - even if it is an empty Pascal string (a single 0 byte).

Remember, any initialization data you specify via the "data" atom is of course added to the data reference, which means the data reference can grow to be quite large. You'll probably only want to use image data (GIF, etc.), text data or music data as initialization data (rather than audio/video data) , as it tends to take up less space.

Back to top

Summary

Data reference extensions in QuickTime 4 give you the ability to associate Mac OS file type, MIME type, initialization data and/or file name information with a given file. This assures QuickTime will interpret your data correctly and as quickly as possible.

References

QuickTime 4 API Documentation, Data Reference Data Type

Letters From the Ice Floe Dispatch #5 "Finding the file type and extension from the MIME type"

QuickTime 4 API Documentation, Graphics Importer

QuickTime 4 API Documentation, QuickTime Atoms

Back to top

Downloadables

	Acrobat version of this Note (K).	Download

Back to top