Technical Note TN2040

Apple Image Capture Camera Module changes for Mac OS X Update 10.1.3

CONTENTS

This Technote describes changes to the Apple Image Capture camera modules for Mac OS X Update 10.1.3.

[Apr 16 2002]

New Hierarchical Tree Structure

Apple's Image Capture Architecture is based on two main elements: objects and properties (check the Image Capture SDK documentation for additional information. Objects may contain a reference to other objects, forming a tree-structure, and objects may own properties.

The camera modules for Apple Image Capture on Mac OS X implemented only a flat (non-hierarchical) object structure. In order to get all the images for a device, it was sufficient to simply call the `ICAGetChildCount` function for the device object (type `ICAObject`) as follows:

Listing 1. Old technique for walking the device tree

```
void ParseObjectHierarchy(ICAObject deviceObject)
{
    ICAGetChildCountPB getChildCountPB;
    OSErr err;

    memset(&getChildCountPB, 0, sizeof(ICAGetChildCountPB));
    getChildCountPB.object = deviceObject;
        /* get child count for this device */
    err = ICAGetChildCount(&getChildCountPB, nil);
    if (err == noErr)
    {
        UInt32 index;

            /* iterate over each child object */
        for (index = 0; index < getChildCountPB.count; ++index)
        {
          /* add your code to process each child object here... */
        }
    }

}
```

With Image Capture for Mac OS X Update 10.1.3, the flat `ICAObject` structure has been changed to a new hierarchical tree structure, so the above code is no longer sufficient to traverse the object hierarchy to locate all the objects for the given device.

To get all the images/movies/audio files for a given device, there are now two options - you can walk the device tree by issuing a series of `ICAGetChildCount/ICAGetNthChild` functions calls, or you can use the `ICACopyObjectPropertyDictionary` function with the device object and obtain both a "data" array and a "tree" dictionary of all the objects.

Listing 2 below contains code showing how to walk the device tree by issuing a series of `ICAGetChildCount/ICAGetNthChild` function calls:

Listing 2. New technique for walking the device tree

```
void ParseObjectHierarchy(ICAObject object)
{
    UInt32 imageCount;
    UInt32 imageCountIndex;

    imageCount = GetNumberOfChildrenForObject (object);
    /* iterate over the object hierarchy for the given parent object */
    for (imageCountIndex = 0; imageCountIndex <
                            imageCount; ++imageCountIndex)
    {
        ICAObject      childObject;
        ICAGetChildCountPB getChildCountPB;
        OSErr          result;

        /* get the next child object */
        childObject = getNthChild(object, imageCountIndex);
        if (childObject == NULL)
            break;

        memset(&getChildCountPB, 0, sizeof(ICAGetChildCountPB));
        getChildCountPB.object = childObject;
        result = ICAGetChildCount(&getChildCountPB, nil);
            /* if child count = 0 we have no further children, so
               let's stop parsing the object hierarchy and grab
               the image data for this object */
        if (getChildCountPB.count == 0)
        {
           /* add your code to process the child object here.... */
        }
        else
          /* this object has children, so let's use the current
             object as the new parent and traverse the new hierarchy */
        {
        ParseObjectHierarchy (childObject);
        }
    }
}
```

Alternately, you can use the `ICACopyObjectPropertyDictionary` function with the device object. The `ICACopyObjectPropertyDictionary` function will return a Mac OS Core Foundation `CFDictionary` containing a "data" array that represents the flattened `ICAObject` structure (use either the `CFArrayGetCount` or `NSDictionary`'s count method to get the number of files).

In addition, with Mac OS X Update 10.1.3 the returned dictionary will now contain a "tree" dictionary that represents the entire hierarchical object tree.

Listing 3 below contains code showing how to get a list of all objects using the "data" array in the dictionary returned by the `ICACopyObjectPropertyDictionary` function:

Listing 3. Obtaining a list of all objects using the "data" array in the dictionary returned by the `ICACopyObjectPropertyDictionary`.

```
OSErr ParseDeviceObjects (ICAObject deviceObject)
{
    OSErr                       err = noErr;
    ICACopyObjectPropertyDictionaryPB   dictionaryPB;
    CFDictionaryRef                 dict = NULL;
    CFArrayRef                  objectArray = NULL;
    CFIndex                     objectCount = 0, index;

    memset(&dictionaryPB, 0, sizeof(ICACopyObjectPropertyDictionaryPB));
    dictionaryPB.object     = deviceObject;
    dictionaryPB.theDict    = &dict;
    err = ICACopyObjectPropertyDictionary(&dictionaryPB, nil);
    if (err != noErr) goto bail;

        /* get 'data' dictionary */
    objectArray = (CFArrayRef)CFDictionaryGetValue(dict, CFSTR("data"));
    if (objectArray ==  NULL) goto bail;

        /* get object count */
    objectCount = CFArrayGetCount(objectArray);
```

```
        /* iterate over each object */
    for (index = 0; index < objectCount; ++index)
    {
        ICAObject       object;
        CFDictionaryRef     dictRef;
        CFNumberRef     theValue;

            /* get the CFDictionaryRef for this object */
        dictRef = (CFDictionaryRef)CFArrayGetValueAtIndex (objectArray,index);

            /* You can use the CFShow function for debugging to
            dump all the key/value pairs for the CFDictionaryRef */
        CFShow(dictRef);

            /* now you can extract any of the values for this
            CFDictionaryRef. As an example, here's how
            to get the value corresponding to the "tsiz"
            key */
        theValue = (CFNumberRef)CFDictionaryGetValue(dictRef, CFSTR("tsiz"));
        if (theValue)
        {
            Boolean gotValue = false;
            SInt32  actualNumber;

            gotValue = CFNumberGetValue(theValue,
                        kCFNumberSInt32Type,&actualNumber);

        }

    /* add your code to extract any additional values here... */
    }

 bail:
    if (dict != NULL)
        CFRelease(dict);

    return err;
}
```

Back to top

Summary

The Apple Image Capture camera modules have been changed with Mac OS X Update 10.1.3, and it is no longer sufficient to simply call the `ICAGetChildCount` function for the device object to obtain a list of all the objects for a given device. Instead, developers will now need to use the techniques described in this note to parse the object hierarchy.

Back to top

References

Mac OS X Core Foundation Collection Services

Downloadables

Acrobat version of this Note (68K). Download

Back to top
