

MAID 3.1

Revision 17

February 02, 2001

Contents

1	Authors	1
2	Concepts	3
2.1	The Big Picture	3
2.2	Abstractions	3
2.3	The Life Of An Object	4
2.4	Double Vision	4
3	Usage	7
3.1	Module Loading	7
3.2	Module Initialization	7
3.3	Capability Enumeration	8
3.4	Reading Array Capabilities	8
3.5	Using Range Capabilities	10
3.6	Capability Groups	10
3.7	Using Vendor Specific Capabilities	10
3.8	Opening Sources and Items	10
3.9	Opening Data Objects	11
3.10	Data Transfer	11
3.11	Saving And Restoring State	12
3.12	Event Notification	13
3.13	User Interface Requests	13
3.14	Asynchronous Module Calls	13
3.15	Module Termination	14
3.16	Module Unloading	14
4	Enumerations	15
4.1	Result Codes	15
4.2	Data Object Types	15
4.3	Data Types	16
4.4	Array Types	16
4.5	Capability Types	17
4.6	Capability Operations	17
4.7	Capability Visibility	17
4.8	Object Types	18
4.9	Events	18
4.10	User Interface Request Types	18
4.11	User Interface Results	18
4.12	Filters	19
4.13	Commands	19
4.14	Capabilities	20
4.15	Color Spaces	22
4.16	Boolean Defaults	22
4.17	Module Types	22
4.18	File Data Types	23
4.19	Flash Modes	23

5	Structures and Types	25
5.1	Word Value	25
5.2	Unsigned Long Value	25
5.3	Parameter Value	25
5.4	Pointer Value	25
5.5	Reference Value	25
5.6	MAID Entry Point Function Pointer	25
5.7	MAID Completion Function Pointer	25
5.8	MAID Data Delivery Function Pointer	26
5.9	MAID Event Notification Function Pointer	26
5.10	MAID Progress Notification Function Pointer	26
5.11	MAID User Interface Request Function Pointer	26
5.12	Callback Definition Structure	26
5.13	Date/Time Structure	26
5.14	Point Structure	27
5.15	Size Structure	27
5.16	Rectangle Structure	27
5.17	String Structure	27
5.18	Array Structure	27
5.19	Range Structure	28
5.20	Capability Information Structure	28
5.21	Object Structure	29
5.22	User Interface Request Structure	29
5.23	Generic Data Delivery Structure	31
5.24	Image Data Delivery Structure	31
5.25	Sound Data Delivery Structure	32
5.26	Enumeration Structure	32
5.27	File Data Delivery Structure	33
6	Result Codes	35
6.1	kNkMAIDResult_NotSupported	35
6.2	kNkMAIDResult_UnexpectedDataType	35
6.3	kNkMAIDResult_ValueOutOfBounds	35
6.4	kNkMAIDResult_BufferSize	35
6.5	kNkMAIDResult_Aborted	35
6.6	kNkMAIDResult_NoMedia	35
6.7	kNkMAIDResult_NoEventProc	35
6.8	kNkMAIDResult_ZombieObject	35
6.9	kNkMAIDResult_NoError	36
6.10	kNkMAIDResult_Pending	36
6.11	kNkMAIDResult_OrphanedChildren	36
6.12	kNkMAIDResult_NoDataProc	36
6.13	kNkMAIDResult_OutOfMemory	36
6.14	kNkMAIDResult_UnexpectedError	36
6.15	kNkMAIDResult_HardwareError	36
6.16	kNkMAIDResult_MissingComponent	36
7	Events	37
7.1	kNkMAIDEvent_AddChild	37
7.2	kNkMAIDEvent_RemoveChild	37
7.3	kNkMAIDEvent_WarmingUp	37
7.4	kNkMAIDEvent_WarmedUp	37
7.5	kNkMAIDEvent_CapChange	37
7.6	kNkMAIDEvent_OrphanedChildren	38

7.7	kNkMAIDEvent_CapChangeValueOnly	38
8	Commands	39
8.1	kNkMAIDCommand_Async	39
8.2	kNkMAIDCommand_Open	40
8.3	kNkMAIDCommand_Close	40
8.4	kNkMAIDCommand_GetCapCount	40
8.5	kNkMAIDCommand_GetCapInfo	41
8.6	kNkMAIDCommand_CapStart	41
8.7	kNkMAIDCommand_CapSet	41
8.8	kNkMAIDCommand_CapGet	42
8.9	kNkMAIDCommand_CapGetDefault	42
8.10	kNkMAIDCommand_CapGetArray	43
8.11	kNkMAIDCommand_Mark	43
8.12	kNkMAIDCommand_AbortToMark	44
8.13	kNkMAIDCommand_Abort	44
8.14	kNkMAIDCommand_EnumChildren	44
8.15	kNkMAIDCommand_GetParent	44
8.16	kNkMAIDCommand_ResetToDefault	45
9	Capabilities	47
9.1	kNkMAIDCapability_AsyncRate	47
9.2	kNkMAIDCapability_ProgressProc	48
9.3	kNkMAIDCapability_EventProc	48
9.4	kNkMAIDCapability_DataProc	48
9.5	kNkMAIDCapability_UIRequestProc	48
9.6	kNkMAIDCapability_IsAlive	49
9.7	kNkMAIDCapability_Children	49
9.8	kNkMAIDCapability_State	49
9.9	kNkMAIDCapability_Name	49
9.10	kNkMAIDCapability_Description	49
9.11	kNkMAIDCapability_Interface	50
9.12	kNkMAIDCapability_DataTypes	50
9.13	kNkMAIDCapability_DateTime	50
9.14	kNkMAIDCapability_StoredBytes	50
9.15	kNkMAIDCapability_Eject	50
9.16	kNkMAIDCapability_Feed	51
9.17	kNkMAIDCapability_Capture	51
9.18	kNkMAIDCapability_Mode	51
9.19	kNkMAIDCapability_Acquire	51
9.20	kNkMAIDCapability_Start	51
9.21	kNkMAIDCapability_Length	52
9.22	kNkMAIDCapability_SampleRate	52
9.23	kNkMAIDCapability_Stereo	52
9.24	kNkMAIDCapability_Samples	52
9.25	kNkMAIDCapability_Filter	52
9.26	kNkMAIDCapability_Prescan	52
9.27	kNkMAIDCapability_AutoFocus	53
9.28	kNkMAIDCapability_AutoFocusPt	53
9.29	kNkMAIDCapability_Focus	53
9.30	kNkMAIDCapability_Coords	53
9.31	kNkMAIDCapability_Resolution	53
9.32	kNkMAIDCapability_Preview	53
9.33	kNkMAIDCapability_Negative	54
9.34	kNkMAIDCapability_ColorSpace	54

9.35	kNkMAIDCapability_Bits	54
9.36	kNkMAIDCapability_Planar	54
9.37	kNkMAIDCapability_Lut	54
9.38	kNkMAIDCapability_Transparency	55
9.39	kNkMAIDCapability_Threshold	55
9.40	kNkMAIDCapability_Pixels	55
9.41	kNkMAIDCapability_ForceScan	55
9.42	kNkMAIDCapability_ForcePrescan	55
9.43	kNkMAIDCapability_ForceAutoFocus	56
9.44	kNkMAIDCapability_NegativeDefault	56
9.45	kNkMAIDCapability_Firmware	56
9.46	kNkMAIDCapability_CommunicationLevel1	56
9.47	kNkMAIDCapability_CommunicationLevel2	57
9.48	kNkMAIDCapability_BatteryLevel	57
9.49	kNkMAIDCapability_FreeBytes	57
9.50	kNkMAIDCapability_FreeItems	57
9.51	kNkMAIDCapability_Remove	58
9.52	kNkMAIDCapability_FlashMode	58
9.53	kNkMAIDCapability_ModuleType	58
9.54	kNkMAIDCapability_AcquireStreamStart	58
9.55	kNkMAIDCapability_AcquireStreamStop	58
9.56	kNkMAIDCapability_AcceptDiskAcquisition	59
9.57	kNkMAIDCapability_Version	59
9.58	kNkMAIDCapability_FilmFormat	59

10 Function Definitions **61**

10.1	MAID Entry Point Function	61
10.2	MAID Completion Function	61
10.3	MAID Data Delivery Function	61
10.4	MAID Event Notification Function	61
10.5	MAID Progress Notification Function	62
10.6	MAID User Interface Request Function	62

11 History **63**

11.1	Changes Since v3.0 Revision 2	63
11.2	Changes Since v3.0 Revision 3	63
11.3	Changes Since v3.0 Revision 4	63
11.4	Changes Since v3.0 Revision 5	64
11.5	Changes Since v3.0 Revision 6	64
11.6	Changes Since v3.0 Revision 7	64
11.7	Changes Since v3.0 Revision 8	64
11.8	Changes Since v3.0 Revision 9	65
11.9	Changes Since v3.0 Revision 10	65
11.10	Changes Since v3.0 Revision 11	65
11.10	Changes Since v3.0 Revision 12	65
11.11	Changes Since v3.0 Revision 13	65
11.12	Changes Since v3.1 Revision 1	66
11.13	Changes Since v3.1 Revision 2	66
11.14	Changes Since v3.1 Revision 3	66
11.15	Changes Since v3.1 Revision 4	66
11.16	Changes Since v3.1 Revision 5	66
11.17	Changes Since v3.1 Revision 6	66
11.18	Changes Since v3.1 Revision 7	66
11.19	Changes Since v3.1 Revision 8	66
11.20	Changes Since v3.1 Revision 9	66

11.21 Changes Since v3.1 Revision 10	66
11.22 Changes Since v3.1 Revision 11	66
11.23 Changes Since v3.1 Revision 12	67
11.24 Changes Since v3.1 Revision 14	67
11.25 Changes Since v3.1 Revision 15	67

1 Authors

2 Concepts

This chapter will introduce the ideas behind MAID. Within this chapter, **bold type** indicates the introduction of terms that are used throughout this document.

2.1 The Big Picture

The MAID specification was created to provide an interface layer between an application and a device driver for devices that hold images, sound and/or video. Throughout MAID, the application side is referred to as the **client** and the device driver side is referred to as the **module**. The client provides all of the user interface and the module provides all of the device communication.

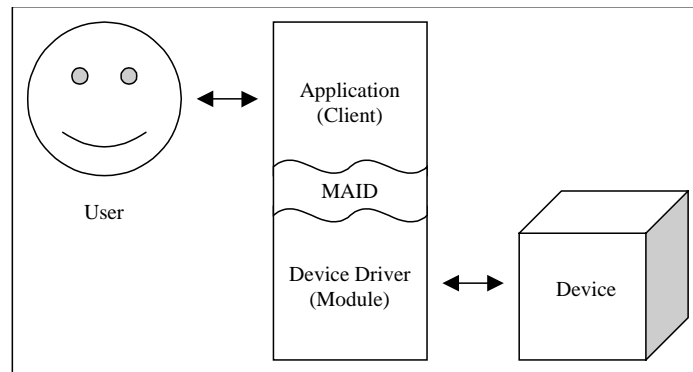


Figure 1

2.2 Abstractions

MAID abstracts the device communication into layers. At each layer, the client opens a view to the module. That view is referred to as an **object**. At the root level is the **module** object, representing the module itself. Next is the **source** object, representing the physical device. Next is the **item** object representing a collection of the deepest level objects, the **data** objects, which represent an image, sound or video. There can be only one of each type of data object within an item.

Throughout this document, “module” is used to refer to the device driver and “module object” is used to refer to the channel that has been opened to the MAID abstraction of the device driver.

As an example, there are two devices available that are supported by a module. The first device has an image and an unrelated sound stored within it. The second device has an image with sound and an unrelated video with sound stored within it. If the client were to open a module object and an object for each of these physical counterparts, the hierarchy of objects would appear as in Figure 2.

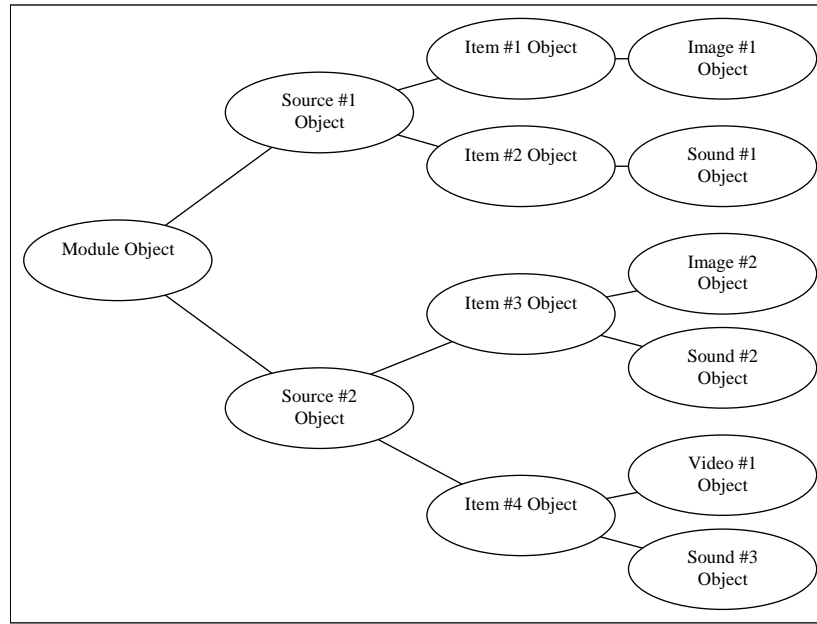


Figure 2

2.3 The Life Of An Object

An object remains open until the client closes it, which the client must do before unloading the module. When a source, item or data object is opened, a **parent** object must be supplied. The object being opened is the **child** object. All settings, known as **capabilities**, are associated with a particular object. The capability settings of the parent object apply to all of it's children.

Normally, while an object is open, it is **alive**. It is no longer alive if the client closes it's parent or the object's physical counterpart disappears. In this **zombie** state, the client can still read some capabilities for the object. Which capabilities are still available depends on what information the module can provide without the object's parent or physical counterpart.

2.4 Double Vision

Since objects are merely views of physical counterparts, more than one object can be opened for one physical counterpart. This is true at each level of the MAID object hierarchy. For example, if the client opens a source twice, both source objects have the same parent module object and access the same physical device as in Figure 3.

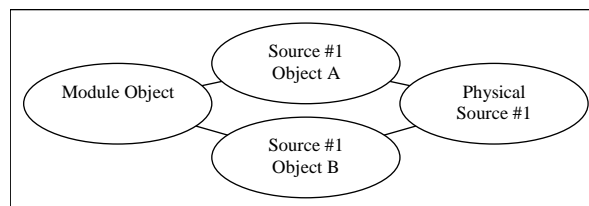


Figure 3

Each of those source objects has it's own settings for the physical object. The module will only allow one of the source objects to have access to the physical source at a time. If the two source objects have a crucial capability set to conflicting values, only the module can decide how to resolve that predicament in a way that is most satisfactory.

This feature is most useful at the data object level. For image type data, one object can be used for a preview acquire while another is used for a final acquire. Another example is gang scanning where multiple originals are placed on a

scanner. For example, the client can have a full area preview object, three closeup preview objects and three final objects. Each of these objects will be referring to the same physical image data coming from the same physical scanner. Obviously if the client started to acquire all seven objects at the same time, the module would scan each one in succession.

3 Usage

To simplify the examples shown here, all calls to the module will be made synchronously. Asynchronous operation is explained below.

3.1 Module Loading

The process of loading a module is system dependent.

For the Windows environment, modules are implemented as DLLs. Each module will have an extension of “.MD3” and must reside somewhere in the directory tree under the “Common Files/MaidMods” directory. The actual name and location of the “Common Files” directory is machine-dependent and may be obtained by querying the appropriate registry entry. Each module will export one function which is the MAID entry point. The client will use LoadLibrary() to load the module and GetProcAddress() to get the address of the MAID entry point. For 32 bit builds, a TWAIN client will be based at 0x01400000 and modules will be based at 0x02400000 or above. This rebasing will shorten load times.

For the Macintosh environment, the module will consist of a Fat file in the System Folder, with the data fork being a CFM library, usable from PowerPC Macs. The resource fork will contain all of the resources necessary to run the module along with a resource of type 'MAID' which will represent the object code needed to run from 68k Macintoshes. On the PowerPC the client will call GetDiskFragment() and pass the result into a UPP descriptor for the MAID entry point. On 68k the client will load the correct code resource, lock the handle, and then typecast the data as a pointer to the MAID entry point.

3.2 Module Initialization

The first step for the client is to have a module structure. It can be either aggregated in a structure or object or allocated from the heap. It should be noted that this example will only be able to open one module at a time because the module structure is a single global variable. (Line 1) A real world client will most likely have more than one module open at once.

The client will initialize the structure by setting the **refClient** member. (Line 9) It will then call the module (Line 12) with a NULL pointer as the object and a pointer to the module structure as the data. The module is open if the command completes successfully. (Line 16) While the object is open, the client will not change the value of the **refClient** member and the module will not change the value of the **refModule** member. No two objects can have the same value for **refClient** or **refModule**.

```
1   NkMAIDObject objModule;
2
3   // open the module synchronously
4   BOOL InitializeMAIDModuleSync( NKREF ref, LPMAIDEntryPointProc pMAIDEntryPoint )
5   {
6       LONG nResult;
7
8       // set the reference
9       objModule.refClient = ref;
10
11      // call the module to open the module
12      nResult = (*pMAIDEntryPoint)( NULL, kNkMAIDCommand_Open, 0, kNkMAIDDataType_ObjectPtr,
13                                  (NKPARAM)&objModule, NULL, 0 );
14
15      // return TRUE if the module successfully opened mimimimi
16      return (nResult == kNkMAIDResult_NoError);
17  }
```

3.3 Capability Enumeration

Once an object has been opened, the capabilities must be enumerated. The client will call the module to get the number of capabilities for that object. (Line 10) If that command completes successfully, the client will allocate memory to hold the capability information. (Line 16) If the memory is allocated successfully, the client will call the module to retrieve the capability information. (Line 21) If the number of capabilities changed between the two calls to the module (Line 24), the memory is released (Line 26) and the process repeats. (Line 33) The capabilities are enumerated if the commands complete successfully. (Line 36)

```

1  // enumerate the capabilities of an object
2  BOOL EnumerateMAIDObjectCapabilities( LPMAIDEntryPointProc pMAIDEntryPoint, LPNkMAIDObject
pObject,
3  ULONG FAR *pulCapCount, LPNkMAIDCapInfo FAR * ppCapArray )
4  {
5      LONG nResult;
6
7      do
8      {
9          // call the module to get the size of the capability array
10         nResult = (*pMAIDEntryPoint)( pObject, kNkMAIDCommand_GetCapCount, 0,
11             kNkMAIDDataType_UnsignedPtr, pulCapCount, NULL, 0 );
12
13         if (nResult == kNkMAIDResult_NoError)
14         {
15             // allocate memory for the array
16             *ppCapArray = (LPNkMAIDCapInfo)malloc( *pulCapCount * sizeof( NkMAIDCapInfo ) );
17
18             if (*ppCapArray != NULL)
19             {
20                 // call the module to get the capability array
21                 nResult = (*pMAIDEntryPoint)( pObject, kNkMAIDCommand_GetCapInfo, *pulCapCount,
22                     kNkMAIDDataType_CapInfoPtr, (NKPARAM)*ppCapArray, NULL, 0 );
23
24                 if (nResult == kNkMAIDResult_BufferSize)
25                 {
26                     free( *ppCapArray );
27                     *ppCapArray = NULL;
28                 }
29             }
30         }
31     }
32     // repeat the process if the number of capabilities changed between the two calls to the
module
33     while (nResult == kNkMAIDResult_BufferSize);
34
35     // return TRUE if the capabilities were successfully enumerated
36     return (nResult == kNkMAIDResult_NoError);
37 }

```

3.4 Reading Array Capabilities

The process of reading array capabilities is similar to reading capability information. The client must check that the capability is an array type and that it supports the *kNkMAIDCommand_CapGetArray* and *kNkMAIDCommand_CapGet* commands (Line 54). The client will call the module to get information about the array capability. (Line 62) If that command completes successfully, the client will allocate memory to hold the array data. (Line 68) If the memory is allocated successfully, the client will call the module to retrieve the array data. (Line 73) If the size of the array data changed between the two calls to the module (Line 76), the memory is released (Line 78) and the process repeats. (Line 85) The array has been read if the commands complete successfully. (Line 88)

```

1  // find the capability
2  BOOL FindMAIDCapability( LPMAIDEntryPointProc pMAIDEntryPoint, LPNkMAIDObject pObject,
3  ULONG ulCapID, LPNkMAIDCapInfo pCapInfo )
4  {
5      LONG nResult;
6      BOOL fRet = FALSE;
7      ULONG ulCapCount;
8      LPNkMAIDCapInfo lpCapArray;
9
10     // make sure we don't free some memory we didn't allocate

```



```

11     lpCapArray = NULL;
12
13     // this function is in the example for capability enumeration
14     if (EnumerateMAIDObjectCapabilities( pMAIDEntryPoint, pObject, &ulCapCount, &lpCapArray))
15     {
16         // make sure we got an array
17         if (lpCapArray != NULL)
18         {
19             ULONG ulIndex;
20
21             // find the capability
22             for (ulIndex=0; ulIndex<ulCapCount; ++ulIndex)
23                 if (lpCapArray[ulIndex].ulID == ulCapID)
24                     break;
25
26             // did we find it?
27             if (ulIndex < ulCapCount)
28             {
29                 fRet = TRUE;
30                 *pCapInfo = lpCapArray[ulIndex];
31             }
32         }
33     }
34
35     // make sure to free memory allocated by EnumerateMAIDObjectCapabilities()
36     if (lpCapArray != NULL)
37         free( lpCapArray );
38
39     return fRet;
40 }
41
42 // enumerate the capabilities of an object
43 BOOL ReadMAIDArrayCapability( LPMAIDEntryPointProc pMAIDEntryPoint, LPNkMAIDObject pObject,
44     ULONG ulCapID, LPNkMAIDArray pArray )
45 {
46     LONG nResult;
47     NkMAIDCapInfo capInfo;
48
49     // get the capability information
50     if (!FindMAIDCapability( pMAIDEntryPoint, pObject, ulCapID, &capInfo ))
51         return FALSE;
52
53     // the capability must be an array, and must support the CapGetArray and CapGet commands
54     if (capInfo->ulType != kNkMAIDCapType_Array ||
55         !(capInfo-> ulOperations & kNkMAIDCapOperation_GetArray) ||
56         !(capInfo-> ulOperations & kNkMAIDCapOperation_Get))
57         return FALSE;
58
59     do
60     {
61         // call the module to get the size of the array data
62         nResult = (*pMAIDEntryPoint)( pObject, kNkMAIDCommand_CapGet, ulCapID,
63             kNkMAIDDataType_ArrayPtr, (NKPARAM)pArray, NULL, 0 );
64
65         if (nResult == kNkMAIDResult_NoError)
66         {
67             // allocate memory for the array
68             pArray->pData = malloc( pArray->ulElements * pArray->wPhysicalBytes );
69
70             if (pArray->pData != NULL)
71             {
72                 // call the module to get the array data
73                 nResult = (*pMAIDEntryPoint)( pObject, kNkMAIDCommand_CapGetArray, ulCapID,
74                     kNkMAIDDataType_ArrayPtr, (NKPARAM)pArray, NULL, 0 );
75
76                 if (nResult == kNkMAIDResult_BufferSize)
77                 {
78                     free( pArray->pData );
79                     pArray->pData = NULL;
80                 }
81             }
82         }
83     }
84     // repeat the process if the array data size changed between the two calls to the module
85     while (nResult == kNkMAIDResult_BufferSize);
86

```

```

87     // return TRUE if the array was successfully read
88     return (nResult == kNkMAIDResult_NoError);
89 }

```

3.5 Using Range Capabilities

3.6 Capability Groups

3.7 Using Vendor Specific Capabilities

3.8 Opening Sources and Items

To open a source or item object, the client first needs, respectively, a module or source object to use as a parent. The client will find the parent's *kNkMAIDCapability_Children* capability. (Line 9) The client will read this array capability (Line 14) and choose an ID of a child object. The client will initialize the child object structure by setting the **refClient** member. (Line 47) It will then call the module (Line 50) with a pointer to the structure to open the child object. It is open if the command completes successfully. (Line 55)

For the sake of this example, the capabilities are enumerated and the child ID array is read every time a child is to be opened. A real-world client would cache both of these to minimize the conversation between the client and the module and increase speed.

```

1  // read the child IDs into an array structure
2  BOOL GetMAIDChildIDs(LPMAIDEntryPointProc pMAIDEntryPoint, LPNkMAIDObject pParentObject,
3  LPNkMAIDArray pchildIDArray )
4  {
5      LONG nResult;
6      NkMAIDCapInfo capInfo;
7
8      // this function is in the example for array capabilities
9      if (!FindMAIDCapability( pMAIDEntryPoint, pParentObject, kNkMAIDCapability_Children,
10 &capInfo ))
11         return FALSE;
12
13     // this function is in the example for array capabilities
14     if (ReadMAIDArrayCapability( pMAIDEntryPoint, pParentObject, kNkMAIDCapability_Children,
15 pchildIDArray ))
16     {
17         // the array must be 32 bit unsigned integers
18         if (pchildIDArray->ulType != kNkMAIDArrayType_Unsigned ||
19             pchildIDArray->wPhysicalSize != 4 || pchildIDArray->wLogicalBits != 32)
20             return FALSE;
21     }
22
23     return TRUE;
24 }
25
26 // open child object
27 BOOL OpenMAIDChild( LPMAIDEntryPointProc pMAIDEntryPoint, LPNkMAIDObject pParentObject,
28 ULONG ulChildIndex, NKREF refChild, LPNkMAIDObject pChildObject )
29 {
30     LONG nResult;
31     BOOL fRet = FALSE;
32     NkMAIDArray childIDArray;
33
34     // make sure we don't free some memory we didn't allocate
35     childIDArray.pData = NULL;
36
37     // get array of child IDs
38     if (GetMAIDChildIDs( pMAIDEntryPoint, pParentObject, &childIDArray ))

```

```

39     {
40         // ulChildIndex must be a valid index
41         if (childIDArray.ulElements > ulChildIndex && childIDArray.pData != NULL)
42         {
43             // get the ID of the child from the array
44             ULONG FAR *pulChildID = (ULONG FAR *)childIDArray.pData;
45
46             // set the reference
47             pChildObject->refClient = refChild;
48
49             // tell the module to open the child
50             nResult = (*pMAIDEntryPoint)( pParentObject, kNkMAIDCommand_Open,
51             pulChildID[ulChildIndex], kNkMAIDDataType_ObjectPtr,
52             (NKPARAM)pChildObject, NULL, 0 );
53
54             // return TRUE if the child was successfully opened
55             fRet = (nResult == kNkMAIDResult_NoError);
56         }
57     }
58
59     // make sure to free memory allocated by ReadMAIDArrayCapability()
60     if (childIDArray.pData != NULL)
61         free( childIDArray.pData );
62
63     return fRet;
64 }

```

3.9 Opening Data Objects

To open a data object, the client first needs an item object to use as a parent. The client will get the data types available from the item (Line 10) and check if the data type it wants is valid (Line 15). The client will initialize the data object structure by setting the **refClient** member. (Line 18) The client will then call the module (Line 21) with a pointer to the data object structure to open the data object. It is open if the command completes successfully. (Line 25)

```

1  // open data object
2  BOOL OpenMAIDDataObject( LPMAIDEntryPointProc pMAIDEntryPoint, LPNkMAIDObject pItemObject,
3  ULONG ulDataObjectType, NKREF refChild, LPNkMAIDObject pDataObject )
4  {
5      LONG nResult;
6      BOOL fRet = FALSE;
7      ULONG ulDataTypes;
8
9      // get the data types available for this item
10     nResult = (*pMAIDEntryPoint)( pItemObject, kNkMAIDCommand_CapGet,
11     kNkMAIDCapability_DataTypes, kNkMAIDDataType_UnsignedPtr,
12     (NKPARAM)( ULONG FAR *)&ulDataTypes, NULL, 0 );
13
14     // make sure we got an answer and that the data type requested is available
15     if (nResult == kNkMAIDResult_NoError && (ulDataTypes & ulDataObjectType) != 0)
16     {
17         // set the reference
18         pDataObject->refClient = refChild;
19
20         // tell the module to open the data object
21         nResult = (*pMAIDEntryPoint)( pItemObject, kNkMAIDCommand_Open,
22         ulDataObjectType, kNkMAIDDataType_ObjectPtr, (NKPARAM)pDataObject, NULL, 0 );
23
24         // return TRUE if the child was successfully opened
25         fRet = (nResult == kNkMAIDResult_NoError);
26     }
27
28     return fRet;
29 }

```

3.10 Data Transfer

This is how you do it.

```

1  // acquire a data object
2  BOOL AcquireMAIDDataObject( LPMAIDEntryPointProc pMAIDEntryPoint, LPNkMAIDObject pDataObject,
3  LPVOID FAR *ppData )
4  {

```

```

5     LONG nResult;
6     BOOL fRet = FALSE;
7     ULONG ulDataSize;
8     NkMAIDCallback cbDataProc;
9
10    // find out how large the data will be - this is different for images, sound and video
11    :
12    :
13
14    if (nResult == kNkMAIDResult_NoError)
15    {
16        // allocate the memory we need
17        *ppData = malloc( ulDataSize );
18
19        if (*ppData != NULL)
20        {
21            // make sure the data delivery callback function gets a pointer to the memory
22            cbDataProc.pProc = (LPNKFUNC)ReceiveMAIDData;
23            cbDataProc.ref = (NKREF)*ppData;
24
25            // set the data delivery callback function
26            nResult = (*pMAIDEntryPoint)( pDataObject, kNkMAIDCommand_CapSet,
27                kNkMAIDCapability_ProgressProc, kNkMAIDDataType_CallbackPtr,
28                (NKPARAM)(LPNkMAIOCallback)&cbDataProc, NULL, 0 );
29
30            if (nResult == kNkMAIDResult_NoError)
31            {
32                // start the acquire
33                nResult = (*pMAIDEntryPoint)( pDataObject, kNkMAIDCommand_CapStart,
34                    kNkMAIDCapability_Acquire, kNkMAIDDataType_Null, NULL, NULL, 0 );
35
36                // return TRUE if the acquire was successfully completed
37                fRet = (nResult == kNkMAIDResult_NoError);
38            }
39        }
40    }
41
42    return fRet;
43 }
44
45 // copy the delivered data
46 LONG ReceiveMAIDData( LPNkMAIDObject pObject, NKREF ref, LPVOID pDataInfo, LPVOID pData )
47 {
48     LPVOID pBuffer = (LPVOID)ref;    // reference value in callback structure
49
50     // interpret the structure pointed to by pDataInfo and copy the
51     // data in pData to a client allocated buffer
52     :
53     :
54
55     return kNkMAIDResult_NoError;
56 }

```

3.11 Saving And Restoring State

Each module, source, item and data object can each have their own state.

To get the current state of an object, the client only needs to read the *kNkMAIDCapability_State* array capability. The data that is retrieved from that capability should be stored verbatim. To restore the state of an object, the client only needs to set the *kNkMAIDCapability_State* array capability with data that was previously read from the object.

```

1     // get the object state
2     BOOL GetMAIDObjectState( LPMAIDEntryPointProc pMAIDEntryPoint, LPNkMAIDObject pObject,
3         LPNkMAIDArray pStateArray )
4     {
5         LONG nResult;
6         NkMAIDCapInfo capInfo;
7
8         // this function is in the example for array capabilities
9         if (!FindMAIDCapability( pMAIDEntryPoint, pParentObject, kNkMAIDCapability_State,
10             &capInfo ))
11             return FALSE;
12
13         // this function is in the example for array capabilities

```

```

14     if (ReadMAIDArrayCapability( pMAIDEntryPoint, pParentObject, kNkMAIDCapability_State,
15         pStateArray ))
16     {
17         // the array must be 32 bit unsigned integers
18         if (pStateArray->ulType != kNkMAIDArrayType_Unsigned ||
19             pStateArray->wPhysicalSize != 1 || pStateArray->wLogicalBits != 8)
20             return FALSE;
21     }
22
23     return TRUE;
24 }
25
26 // Set the object state
27 BOOL SetMAIDObjectState( LPMAIDEntryPointProc pMAIDEntryPoint, LPNkMAIDObject pObject,
28     LPNkMAIDArray pStateArray )
29 {
30     LONG nResult;
31     NkMAIDCapInfo capInfo;
32
33     // this function is in the example for array capabilities
34     if (!FindMAIDCapability( pMAIDEntryPoint, pParentObject, kNkMAIDCapability_State,
35         &capInfo ))
36         return FALSE;
37
38     // set the state
39     nResult = (*pMAIDEntryPoint)( pObject, kNkMAIDCommand_CapSet,
40         kNkMAIDCapability_State, kNkMAIDDataType_ArrayPtr,
41         (NKPARRAY)pStateArray, NULL, 0 );
42
43     return (nResult == kNkMAIDResult_NoError);
44 }

```

3.12 Event Notification

3.13 User Interface Requests

3.14 Asynchronous Module Calls

For the sake of the example, we will be closing an object. The client will call the module with a pointer to a completion function. (Line 39) If the module can execute the command asynchronously, the module will return the *kNkMAIDResult_Pending* result code immediately. In this example, the client waits for the close command to complete by continuously calling the module with the *kNkMAIDCommand_Async* command. The client can direct this command to a certain object to give it priority (Line 20) or let the module decide what to do. (Line 26) The client will check the result code (Line 10) which will be set by the completion callback function. (Line 50) If the module processes the command synchronously, the module will call the completion function before returning and the wait loop (Lines 10-27) will never be executed. The object is closed if the command completes successfully. (Line 31)

```

1  BOOL CloseMAIDObject( LPMAIDEntryPointProc pMAIDEntryPoint, LPNkMAIDObject pObject )
2  {
3      LONG nResult;
4
5      // call the module asynchronously
6      if (CallMAIDAsync( pMAIDEntryPoint, pObject, kNkMAIDCommand_Close, 0, kNkMAIDDataType_NULL, 0,
7          &nResult ))
8      {
9          // loop while processing the command
10         while (nResult == kNkMAIDResult_Pending)
11         {
12             // respond to user interface items or perform other non-MAID operations
13             // ...
14
15             // give a single threaded module a chance to call the callback - the client can
16             // direct the async command at an object or let the module choose what object
17             // it should be directed to

```

```

18
19      // direct the module to process asynchronous commands only for this object
20      (*pMAIDEntryPoint)( pObject, kNkMAIDCommand_Async, 0, kNkMAIDDataType_Null, 0,
21                          NULL, 0 );
22
23      // .. OR ..
24
25      // let the module process asynchronous commands for any object
26      (*pMAIDEntryPoint)( NULL, kNkMAIDCommand_Async, 0, kNkMAIDDataType_Null, 0, NULL, 0 );
27  }
28  }
29
30  // return TRUE if the object was closed
31  return (nResult == kNkMAIDResult_NoError);
32  }
33
34  // call the module asynchronously
35  BOOL CallMAIDAsync( LPMAIDEntryPointProc pMAIDEntryPoint, LPNkMAIDObject pObject, ULONG
ulCommand,
36                     ULONG ulParam, ULONG ulDataType, NKPARAM data, LONG FAR *pnResult )
37  {
38      // call the module
39      *pnResult = (*pMAIDEntryPoint)( pObject, ulCommand, ulParam, ulDataType, data,
40                                     SetMAIDResult, (NKREF)pnResult );
41
42      // return TRUE if the command was started
43      return (nResult == kNkMAIDResult_NoError || nResult == kNkMAIDResult_Pending);
44  }
45
46  // save the result of the command in the reference
47  void SetMAIDResult( LPMAIDObject pObject, ULONG ulCommand, ULONG ulParam, ULONG ulDataType,
48                     NKPARAM data, NKREF refComplete, LONG nResult )
49  {
50      *((LONG FAR *)refComplete) = nResult;
51  }

```

3.15 Module Termination

To terminate a module, the client will close all objects. When closing each object, the module will abort any commands in progress, call the progress callback with **ulDone** equal to **ulTotal** if the callback is present, call the completion function and set the **refModule** member of the NkMAIDObject structure to NULL so the object cannot be used by the client by mistake.

3.16 Module Unloading

The process of unloading a module is system dependent.

For the Windows environment, the client will use FreeLibrary() to unload the module DLL.

For the Macintosh environment, the client will call CloseConnection on the PowerPC to close out the CFM library. On the 68k platform, the client will simply unlock the memory block holding the module code, and dispose of it.

4 Enumerations

4.1 Result Codes

```
enum eNkMAIDResult
{
    // these values are errors
    kNkMAIDResult_NotSupported = -127,
    kNkMAIDResult_UnexpectedDataType,
    kNkMAIDResult_ValueOutOfBounds,
    kNkMAIDResult_BufferSize,
    kNkMAIDResult_Aborted,
    kNkMAIDResult_NoMedia,
    kNkMAIDResult_NoEventProc,
    kNkMAIDResult_NoDataProc,
    kNkMAIDResult_ZombieObject,
    kNkMAIDResult_OutOfMemory,
    kNkMAIDResult_UnexpectedError,
    kNkMAIDResult_HardwareError,
    kNkMAIDResult_MissingComponent,

    kNkMAIDResult_NoError = 0,

    // these values are warnings
    kNkMAIDResult_Pending,
    kNkMAIDResult_OrphanedChildren,

    kNkMAIDResult_VendorBase = +127
};
```

The module will deliver one of these values to the **nResult** parameter of the client's completion callback function and return the same value from the entry point. Errors will have negative values.

4.2 Data Object Types

```
enum eNkMAIDDataObjType
{
    kNkMAIDDataObjType_Image           = 0x00000001,
    kNkMAIDDataObjType_Sound           = 0x00000002,
    kNkMAIDDataObjType_Video           = 0x00000004,
    kNkMAIDDataObjType_Thumbnail       = 0x00000008,
    kNkMAIDDataObjType_File            = 0x00000010
};
```

The module will use one or more of these values to indicate what types of data a module or source can produce and what types of data are available for a specific item. See the description of *kNkMAIDCapability_DataTypes* for more information.

4.3 Data Types

```
enum eNkMAIDDataType
{
    kNkMAIDDataType_Null = 0,
    kNkMAIDDataType_Boolean,           // passed by value, set only
    kNkMAIDDataType_Integer,           // signed 32 bit int, passed by value, set only
    kNkMAIDDataType_Unsigned,          // unsigned 32 bit int, passed by value, set only
    kNkMAIDDataType_BooleanPtr,        // pointer to single byte boolean value(s)
    kNkMAIDDataType_IntegerPtr,         // pointer to signed 4 byte value(s)
    kNkMAIDDataType_UnsignedPtr,        // pointer to unsigned 4 byte value(s)
    kNkMAIDDataType_FloatPtr,           // pointer to DOUB_P value(s)
    kNkMAIDDataType_PointPtr,           // pointer to NkMAIDPoint structure(s)
    kNkMAIDDataType_SizePtr,            // pointer to NkMAIDSize structure(s)
    kNkMAIDDataType_RectPtr,            // pointer to NkMAIDRect structure(s)
    kNkMAIDDataType_StringPtr,          // pointer to NkMAIDString structure(s)
    kNkMAIDDataType_DateTimePtr,        // pointer to NkMAIDDateTime structure(s)
    kNkMAIDDataType_CallbackPtr,        // pointer to NkMAIDCallback structure(s)
    kNkMAIDDataType_RangePtr,           // pointer to NkMAIDRange structure(s)
    kNkMAIDDataType_ArrayPtr,           // pointer to NkMAIDArray structure(s)
    kNkMAIDDataType_EnumPtr,            // pointer to NkMAIDEnum structure(s)
    kNkMAIDDataType_ObjectPtr,          // pointer to NkMAIDObject structure(s)
    kNkMAIDDataType_CapInfoPtr,         // pointer to NkMAIDCapInfo structure(s)
    kNkMAIDDataType_GenericPtr          // pointer to some value
};
```

The client will pass one of these values to the **ulDataType** parameter of the entry point to indicate how the **data** parameter will be interpreted by the module.

4.4 Array Types

```
enum eNkMAIDArrayType
{
    kNkMAIDArrayType_Boolean = 0,      // 1 byte per element
    kNkMAIDArrayType_Integer,           // signed value that is 1, 2 or 4 bytes per element
    kNkMAIDArrayType_Unsigned,          // unsigned value that is 1, 2 or 4 bytes per element
    kNkMAIDArrayType_Float,             // DOUB_P elements
    kNkMAIDArrayType_Point,             // NkMAIDPoint structures
    kNkMAIDArrayType_Size,              // NkMAIDSize structures
    kNkMAIDArrayType_Rect,              // NkMAIDRect structures
    kNkMAIDArrayType_PackedString,      // packed array of strings
    kNkMAIDArrayType_String,            // NkMAIDString structures
    kNkMAIDArrayType_DateTime           // NkMAIDDateTime structures
};
```

The module will set one of these values in the **ulType** member of the NkMAIDArray structure to indicate how the data of the array should be interpreted. See the description of the NkMAIDArray structure for more information.

4.5 Capability Types

```
enum eNkMAIDCapType
{
    kNkMAIDCapType_Process = 0,           // a process that can be started
    kNkMAIDCapType_Boolean,              // single byte boolean value
    kNkMAIDCapType_Integer,               // signed 4 byte value
    kNkMAIDCapType_Unsigned,              // unsigned 4 byte value
    kNkMAIDCapType_Float,                 // DOUB_P value
    kNkMAIDCapType_Point,                 // NkMAIDPoint structure
    kNkMAIDCapType_Size,                  // NkMAIDSize structure
    kNkMAIDCapType_Rect,                  // NkMAIDRect structure
    kNkMAIDCapType_String,                // NkMAIDString structure
    kNkMAIDCapType_DateTime,              // NkMAIDDateTime structure
    kNkMAIDCapType_Callback,              // NkMAIDCallback structure
    kNkMAIDCapType_Array,                  // NkMAIDArray structure
    kNkMAIDCapType_Enum,                  // NkMAIDEnum structure
    kNkMAIDCapType_Range,                  // NkMAIDRange structure
    kNkMAIDCapType_Generic,               // generic pointer
    kNkMAIDCapType_BoolDefault            // NkMAIDBooleanDefault structure
};
```

The module will set one of these values in the **ulType** member of the NkMAIDCapInfo structure to indicate what type of information is represented. See the Capabilities chapter and the description of the NkMAIDCapInfo structure for more information.

4.6 Capability Operations

```
enum eNkMAIDCapOperations
{
    kNkMAIDCapOperation_Start             = 0x0001,
    kNkMAIDCapOperation_Get               = 0x0002,
    kNkMAIDCapOperation_Set               = 0x0004,
    kNkMAIDCapOperation_GetArray,         = 0x0008,
    kNkMAIDCapOperation_GetDefault        = 0x0010
};
```

The module will set one of more of these values in the **ulOperations** member of the NkMAIDCapInfo structure to indicate what operations are permitted on a particular capability. See the Capabilities chapter and the description of the NkMAIDCapInfo structure for more information.

4.7 Capability Visibility

```
enum eNkMAIDCapVisibility
{
    kNkMAIDCapVisibility_Hidden           = 0x0001,
    kNkMAIDCapVisibility_Advanced         = 0x0002,
    kNkMAIDCapVisibility_Vendor           = 0x0004,
    kNkMAIDCapVisibility_Group            = 0x0008,
    kNkMAIDCapVisibility_GroupMember      = 0x0010,
    kNkMAIDCapVisibility_Invalid          = 0x0020
};
```

The module will set one or more of these values in the **ulVisibility** member of the NkMAIDCapInfo structure to indicate what level of visibility a particular capability has. See the Capabilities chapter and the description of the NkMAIDCapInfo structure for more information.

4.8 Object Types

```
enum eNkMAIDObjectType
{
    kNkMAIDObjectType_Module = 1,
    kNkMAIDObjectType_Source,
    kNkMAIDObjectType_Item,
    kNkMAIDObjectType_DataObj
};
```

The module will set one of these values in the **ulType** member of the `NkMAIDObject` structure to indicate what type of object is represented.

4.9 Events

```
enum eNkMAIDEvent
{
    kNkMAIDEvent_AddChild,
    kNkMAIDEvent_RemoveChild,
    kNkMAIDEvent_WarmingUp,
    kNkMAIDEvent_WarmedUp,
    kNkMAIDEvent_CapChange,
    kNkMAIDEvent_OrphanedChildren,
    kNkMAIDEvent_CapChangeValueOnly
};
```

The module will deliver one of these values to the **ulEvent** parameter of the client's event callback function to indicate what event has occurred.

4.10 User Interface Request Types

```
enum eNkMAIDUIRequestType
{
    kNkMAIDUIRequestType_Ok,
    kNkMAIDUIRequestType_OkCancel,
    kNkMAIDUIRequestType_YesNo,
    kNkMAIDUIRequestType_YesNoCancel,
};
```

When the module calls the client's user interface callback function, the **ulType** member of the `NkMAIDUIRequestInfo` structure will be set to one of these values. The user will be presented with the choices specified by the value.

4.11 User Interface Results

```
enum eNkMAIDUIRequestResult
{
    kNkMAIDUIRequestResult_None,
    kNkMAIDUIRequestResult_Ok,
    kNkMAIDUIRequestResult_Cancel,
    kNkMAIDUIRequestResult_Yes,
    kNkMAIDUIRequestResult_No
};
```

When the module calls the client's user interface callback function, the **ulDefault** member of the `NkMAIDUIRequestInfo` structure will be set to one of these values. The value will indicate which button should be highlighted by default. The client's user interface callback function will return one of these values depending on which button the user presses. The *kNkMAIDEventResult_None* value can only be returned if the **fSync** member of the `NkMAIDUIRequestInfo` structure is FALSE.

4.12 Filters

```
enum eNkMAIDFilter
{
    kNkMAIDFilter_White,
    kNkMAIDFilter_Infrared,
    kNkMAIDFilter_Red,
    kNkMAIDFilter_Green,
    kNkMAIDFilter_Blue,
    kNkMAIDFilter_Ultraviolet
};
```

The module will use one or more of these values in the `kNkMAIDCapability_Filter` capability. See the Capabilities chapter for more information.

4.13 Commands

```
enum eNkMAIDCommand
{
    kNkMAIDCommand_Async,           // process asynchronous operations
    kNkMAIDCommand_Open,            // opens a child object
    kNkMAIDCommand_Close,           // closes a previously opened object
    kNkMAIDCommand_GetCapCount,      // get number of capabilities of an object
    kNkMAIDCommand_GetCapInfo,      // get capabilities of an object
    kNkMAIDCommand_CapStart,        // starts capability
    kNkMAIDCommand_CapSet,          // set value of capability
    kNkMAIDCommand_CapGet,          // get value of capability
    kNkMAIDCommand_CapGetDefault,    // get default value of capability
    kNkMAIDCommand_CapGetArray,     // get data for array capability
    kNkMAIDCommand_Mark,            // insert mark in the command queue
    kNkMAIDCommand_AbortToMark,      // abort asynchronous commands to mark
    kNkMAIDCommand_Abort,           // abort current asynchronous command
    kNkMAIDCommand_EnumChildren,     // requests 'add' events for all child IDs
    kNkMAIDCommand_GetParent,        // gets previously opened parent for object
    kNkMAIDCommand_ResetToDefault    // resets all capabilities to their default value
};
```

The client will pass one of these values to the **ulCommand** parameter of the MAID entry point to indicate what operation the module should perform. These commands are explained in detail in the Commands chapter.

4.14 Capabilities

```

enum eNkMAIDCapability
{
    kNkMAIDCapability_AsyncRate = 1, // milliseconds between idle async calls
    kNkMAIDCapability_ProgressProc, // callback during lengthy operation
    kNkMAIDCapability_EventProc, // callback when event occurs
    kNkMAIDCapability_DataProc, // callback to deliver data
    kNkMAIDCapability_UIRequestProc, // callback to show user interface

    kNkMAIDCapability_IsAlive, // FALSE if object is removed or parent closed
    kNkMAIDCapability_Children, // IDs of children objects
    kNkMAIDCapability_State, // current state of the object
    kNkMAIDCapability_Name, // a string representing the name of the object
    kNkMAIDCapability_Description, // a string describing the object
    kNkMAIDCapability_Interface, // a string describing the interface to a device
    kNkMAIDCapability_DataTypes, // what data types are supported or available
    kNkMAIDCapability_DateTime, // date associated with an object
    kNkMAIDCapability_StoredBytes, // read only size of object as stored on device

    kNkMAIDCapability_Eject, // ejects media from a device
    kNkMAIDCapability_Feed, // feeds media into a device
    kNkMAIDCapability_Capture, // captures new item from the source
    kNkMAIDCapability_MediaPresent, // TRUE if item has physical media to acquire

    kNkMAIDCapability_Mode, // mode of this object
    kNkMAIDCapability_Acquire, // begins the acquisition of the object
    kNkMAIDCapability_ForceScan, // If FALSE, unnecessary scans can be eliminated

    kNkMAIDCapability_Start, // start offset (in seconds) of the object
    kNkMAIDCapability_Length, // length (in seconds) of the object
    kNkMAIDCapability_SampleRate, // sampling rate (in samples/sec) of the object
    kNkMAIDCapability_Stereo, // mono or stereo
    kNkMAIDCapability_Samples, // given current state, read only number of samples

    kNkMAIDCapability_Filter, // selects the filter for the light source
    kNkMAIDCapability_Prescan, // performs a prescan
    kNkMAIDCapability_ForcePrescan, // If FALSE, unnecessary prescans can be eliminated
    kNkMAIDCapability_AutoFocus, // sets the focus automatically
    kNkMAIDCapability_ForceAutoFocus, // If FALSE, unnecessary autofocus can be eliminated
    kNkMAIDCapability_AutoFocusPt, // sets the position to focus upon
    kNkMAIDCapability_Focus, // sets the focus
    kNkMAIDCapability_Coords, // rectangle of object in device units
    kNkMAIDCapability_Resolution, // resolution of object (in pixels/inch)
    kNkMAIDCapability_Preview, // preview or final acquire
    kNkMAIDCapability_Negative, // negative or positive original
    kNkMAIDCapability_ColorSpace, // colorspace for image delivery
    kNkMAIDCapability_Bits, // bits per color
    kNkMAIDCapability_Planar, // interleaved or planar data transfer
    kNkMAIDCapability_Lut, // LUT(s) for object
    kNkMAIDCapability_Transparency, // light path of the original
    kNkMAIDCapability_Threshold, // threshold level for lineart images
    kNkMAIDCapability_Pixels, // given current state, read only size of image

    kNkMAIDCapability_NegativeDefault, // Default value for Negative capability
    kNkMAIDCapability_Firmware, // Firmware version number

    kNkMAIDCapability_CommunicationLevel1, // Communication method
    kNkMAIDCapability_CommunicationLevel2, // Communication method
    kNkMAIDCapability_BatteryLevel, // Battery Level in device
    kNkMAIDCapability_FreeBytes, // Free bytes in device
    kNkMAIDCapability_FreeItems, // Free items in device
    kNkMAIDCapability_Remove, // Delete an object
    kNkMAIDCapability_FlashMode, // Flash mode
    kNkMAIDCapability_ModuleType, // Module type
    kNkMAIDCapability_AcquireStreamStart, // Start a stream acquisition
    kNkMAIDCapability_AcquireStreamStop, // Stop a stream acquisition
    kNkMAIDCapability_AcceptDiskAcquisition, // Allow acquisitions to use disk transfer
    kNkMAIDCapability_Version, // MAID version

```

```

    kNkMAIDCapability_FilmFormat,           // Film format (35mm, 6*6 etc)
    kNkMAIDCapability_TotalBytes,          // Total bytes in device storage

    kNkMAIDCapability_VendorBase = 0x8000 // vendor supplied capabilities start here
};

```

The module will use these values in the **ulID** member of **NkMAIDCapInfo** structures to indicate what capabilities it provides to the client. The client will pass one of these values to the **ulParam** parameter of the MAID entry point to indicate what capability to perform the command upon.

The module writer can define capabilities unique to their device. The client will allow the user to interact with those capabilities in a generic manner.

These capabilities are explained in detail in the Capabilities chapter.

4.15 Color Spaces

```

enum eNkMAIDColorSpace
{
    kNkMAIDColorSpace_LineArt,
    kNkMAIDColorSpace_Grey,
    kNkMAIDColorSpace_RGB,
    kNkMAIDColorSpace_sRGB,
    kNkMAIDColorSpace_CMYK,
    kNkMAIDColorSpace_Lab,
    kNkMAIDColorSpace_LCH,
    kNkMAIDColorSpace_AppleRGB,
    kNkMAIDColorSpace_ColorMatchRGB,
    kNkMAIDColorSpace_NTSCRGB,
    kNkMAIDColorSpace_BruceRGB,
    kNkMAIDColorSpace_AdobeRGB,
    kNkMAIDColorSpace_CIERGB,
    kNkMAIDColorSpace_AdobeWideRGB,
    kNkMAIDColorSpace_AppleRGB_Compensated
};

```

The client will use these values in the **kNkMAIDCapability_ColorSpace** capability. See the Capabilities chapter for more information.

4.16 Boolean Defaults

```

enum eNkMAIDBooleanDefault
{
    kNkMAIDBooleanDefault_True,
    kNkMAIDBooleanDefault_False
};

```

The client will use these values with the **kNkMAIDCapability_NegativeDefault** capability. See the Capabilities chapter for more information.

4.17 Module Types

```

enum eNkMAIDModuleTypes
{
    kNkMAIDModuleType_Scanner      = 0x0001,
    kNkMAIDModuleType_Camera       = 0x0002
};

```

The module will return one of more of these values in the **kNkCapability_ModuleType** capability. This will help the client determine if this module should be used, or perhaps which user interface to display.

4.18 File Data Types

```
enum eNkMAIDFileDataTypes
{
    kNkMAIDFileDataType_NotSpecified,
    kNkMAIDFileDataType_JPEG,
    kNkMAIDFileDataType_TIFF,
    kNkMAIDFileDataType_FlashPix,
    kNkMAIDFileDataType_NIF,
    kNkMAIDFileDataType_QuickTime,
    kNkMAIDFileDataType_UserType = 0x100
};
```

The module will use these values in the `NkMAIDFileInfo` structure when sending file data to the client in response to the `kNkMAIDCapability_Acquire` capability.

4.19 Flash Modes

```
enum eNkMAIDFlashMode
{
    kNkMAIDFlashMode_FrontCurtain,
    kNkMAIDFlashMode_RearCurtain,
    kNkMAIDFlashMode_SlowSync,
    kNkMAIDFlashMode_RedEyeReduction,
    kNkMAIDFlashMode_SlowSyncRedEyeReduction,
    kNkMAIDFlashMode_SlowSyncRearCurtain
};
```

The client will use these values in the `kNkMAIDCapability_FlashMode` capability. See the Capabilities chapter for more information.

5 Structures and Types

5.1 Word Value

```
typedef unsigned short WORD;
```

This definition may be implementation dependent with the only requirement that it must be appropriate for 16 bit unsigned integers.

5.2 Unsigned Long Value

```
typedef unsigned long ULONG;
```

This definition may be implementation dependent with the only requirement that it must be appropriate for 32 bit unsigned integers.

5.3 Parameter Value

```
typedef ULONG NKPARAM;
```

This definition may be implementation dependent with the only requirement that it must be appropriate for pointers to objects and 32 bit integers.

5.4 Pointer Value

```
typedef void FAR *LPVOID;
```

This definition may be implementation dependent with the only requirement that it must be appropriate for pointers to objects.

5.5 Reference Value

```
typedef LPVOID NKREF;
```

This definition may be implementation dependent with the only requirement that it must be appropriate for pointers to objects. This type will be used in structures where the client wishes to associate the structure with another structure or object. This is also used by callback functions.

5.6 MAID Entry Point Function Pointer

```
typedef LONG (FAR *LPMAIDEntryPointProc)( LPNkMAIDObject, ULONG, ULONG, ULONG, NKPARAM, LPMAIDCompletionProc, NKREF );
```

5.7 MAID Completion Function Pointer

```
typedef void (FAR *LPMAIDCompletionProc)( LPNkMAIDObject, ULONG, ULONG, ULONG, NKPARAM, NKREF, LONG );
```

5.8 MAID Data Delivery Function Pointer

```
typedef LONG (FAR *LPMAIDDataProc)( NKREF, LPVOID, LPVOID );
```

5.9 MAID Event Notification Function Pointer

```
typedef void (FAR *LPMAIDEventProc)( NKREF, ULONG, NKPARAM );
```

5.10 MAID Progress Notification Function Pointer

```
typedef void (FAR *LPMAIDProgressProc)( ULONG, ULONG, NKREF, ULONG, ULONG );
```

5.11 MAID User Interface Request Function Pointer

```
typedef ULONG (FAR *LPMAIDUIRequestProc)( NKREF, LPNkMAIDUIRequestInfo );
```

5.12 Callback Definition Structure

```
typedef struct tagNkMAIDCallback
{
    LPNKFUNC      pProc;
    NKREF         refProc;
} NkMAIDCallback, FAR* LPNkMAIDCallback;
```

This structure is used to describe a callback function. The **pProc** member points to the function and the **refProc** member is used by the callback for its own purposes. Usually, **refProc** is used by the client to point to an object or structure.

5.13 Date/Time Structure

```
typedef struct tagNkMAIDDateTime
{
    WORD nYear;           // ie 1997, 1998
    WORD nMonth;          // 1-12 = Jan-Dec
    WORD nDay;            // 1-31
    WORD nHour;           // 0-23
    WORD nMinute;         // 0-59
    WORD nSecond;         // 0-59
    ULONG nSubsecond;     // Module dependent
} NkMAIDDateTime, FAR* LPNkMAIDDateTime;
```

Each individual module can decide how to interpret the **nSubsecond** member. For example, if several pictures were taken within one second, they could be assigned sequential numbers starting with zero. Alternatively, it could be implemented as milliseconds.

5.14 Point Structure

```
typedef struct tagNkMAIDPoint
{
    LONG    x;
    LONG    y;
} NkMAIDPoint, FAR* LPNkMAIDPoint;
```

5.15 Size Structure

```
typedef struct tagNkMAIDSize
{
    LONG    w;
    LONG    h;
} NkMAIDSize, FAR* LPNkMAIDSize;
```

5.16 Rectangle Structure

```
typedef struct tagNkMAIDRect
{
    LONG    x;        // left coordinate
    LONG    y;        // top coordinate
    ULONG    w;        // width
    ULONG    h;        // height
} NkMAIDRect, FAR* LPNkMAIDRect;
```

5.17 String Structure

```
typedef struct tagNkMAIDString
{
    SCHAR    str[256];    // allows a 255 character null terminated string
} NkMAIDString, FAR* LPNkMAIDString;
```

The string must be null terminated. Using this structure, the maximum length string that can be transferred is 255 characters long. Consider using the `NkMAIDArray` structure with the **ulType** member set to `kNkMAIDArrayType_PackedString` for longer strings.

5.18 Array Structure

```
typedef struct tagNkMAIDArray
{
    ULONG        ulType;                // one of eNkMAIDArrayType
    ULONG        ulElements;            // total number of elements
    ULONG        ulDimSize1;            // size of first dimension
    ULONG        ulDimSize2;            // size of second dimension, zero for 1 dim
    ULONG        ulDimSize3;            // size of third dimension, zero for 1 or 2 dim
    WORD        wPhysicalBytes;         // bytes per element
    WORD        wLogicalBits;           // must be <= wPhysicalBytes * 8
    LPVOID        pData;                // allocated by the client
} NkMAIDArray, FAR* LPNkMAIDArray;
```

The `NkMAIDArray` structure allows an array to be transferred through the MAID interface. The client will always allocate the memory. It is the responsibility of the receiver of the data to interpret the data properly. The size of **pData** in bytes should always be **ulElements** times **wPhysicalBytes**.

Two and three dimensional arrays may be transferred by setting **ulDimSize1**, **ulDimSize2** and **ulDimSize3**. For a two dimensional array of 20 rows of 10 elements each, **ulDimSize1** will be 10, **ulDimSize2** will be 20 and **ulElements** will be 200. If there are five of those arrays, **ulDimSize1** will be 10, **ulDimSize2** will be 20, **ulDimSize3** will be 5 and **ulElements** will be 1000.

When **ulType** is *kNkMAIDArrayType_Integer* or *kNkMAIDArrayType_Unsigned*, the sender may specify that although the integer values in **pData** are two bytes each (**wPhysicalBytes**), the receiver should interpret them as 10-bit values (**wLogicalBits**). The **wLogicalBits** member is ignored for other types.

When **ulType** is *kNkMAIDArrayType_PackedString*, **pData** will point to a packed list of null terminated strings, **ulElements** will be the total length in bytes of the data including the terminating null bytes, **ulDimSize1** will be the number of strings, **ulDimSize2** will be zero and **wPhysicalBytes** will be one.

5.19 Range Structure

```
typedef struct tagNkMAIDRange
{
    DOUB_P    lfValue;
    DOUB_P    lfDefault;
    ULONG     ulValueIndex;        // zero-based index
    ULONG     ulDefaultIndex;     // zero-based index
    DOUB_P    lfLower;
    DOUB_P    lfUpper;
    ULONG     ulSteps;            // zero for infinite range, otherwise must be >= 2
} NkMAIDRange, FAR* LPNkMAIDRange;
```

This structure is to implement capabilities with a numerical range of values (0-100, -5.0 to +5.0, etc.) The lower and upper limits will be in **lfLower** and **lfUpper** respectively.

If any value from **lfLower** to **lfUpper** is allowed, **lfValue** will be the current value, **lfDefault** will be the default value and **ulSteps** will be zero. In this case, **ulValueIndex** and **ulDefaultIndex** are not used.

If only discrete steps are allowed, **ulValueIndex** will be the index of the current step, **ulDefaultIndex** will be the index of the default step and **ulSteps** will be the number of equally spaced steps including the lower and upper limits. There must be at least two discrete steps. In this case, **lfValue** and **lfDefault** are not used.

There are two ways to set the value of a range. The client can send a pointer to a NkMAIDRange structure with a new value in **lfValue** or **ulValueIndex**. For ranges with discrete steps, the client can send an unsigned integer which the module will accept as the index of the value. In that case, zero is the lower limit and one less than **ulSteps** is the upper limit.

5.20 Capability Information Structure

```
typedef struct tagNkMAIDCapInfo
{
    ULONG     ulID;                // one of eNkMAIDCapability or vendor specified
    ULONG     ulType;              // one of eNkMAIDCapabilityType
    ULONG     ulVisibility;        // eNkCapVisibility bits
    ULONG     ulOperations;        // eNkCapOperations bits
    SCHAR     szDescription[256]; // text describing the capability
} NkMAIDCapInfo, FAR* LPNkMAIDCapInfo;
```

Each of the module, source, item, image and sound objects have their own capabilities. Use the *kNkMAIDCommand_GetCapInfo* to retrieve an array of these structures. Each has a unique identifier value in **ulID**. The ID can be one of the eNkMAIDCapability values or a vendor specific value.

5.21 Object Structure

```
typedef struct tagNkMAIDObject
{
    ULONG        ulType;           // one of eNkMAIDObjectType
    ULONG        ulID;
    NKREF        refClient;
    NKREF        refModule;
} NkMAIDObject, FAR* LPNkMAIDObject;
```

This structure is able to represent the module, source, item, image and sound objects that pass between the MAID client and module.

To open an object, the client will allocate the memory required and fill **refClient** with whatever value it needs. The client then calls the module to open the object. During that call, the module will set **ulType** to the appropriate value, **ulID** to the ID of the object and **refModule** to whatever value it needs. While the object is open, the values of **refClient** and **refModule** will not change.

If a module, source, item, image or sound is opened a second time, the module will maintain a second set of capability values for the new object. Both NkMAIDObject structures will have the same **ulID** value. The module will be able to differentiate the first and second instances by the value of the **refModule** member. The client will be able to differentiate the two by the value of the **refClient** member.

5.22 User Interface Request Structure

```
typedef struct tagNkMAIDUIRequestInfo
{
    ULONG        ulType;           // one of eNkMAIDUIRequestType
    ULONG        ulDefault;        // default value - one of eNkMAIDUIRequestResult
    BOOL         fSync;            // TRUE if user must respond before returning
    char FAR *   lpPrompt;         // NULL terminated text to show to user
    char FAR *   lpDetail;         // NULL terminated text indicating more detail
    LPNkMAIDObject pObject;        // Target Object for data element
    NKPARAM      data;             // Pointer to an NkMAIDArray structure
} NkMAIDUIRequestInfo, FAR* LPNkMAIDUIRequestInfo;
```

When the module wants to notify the user of some event or query a response from the user, it will call the client's user interface function with **pUIRequest** set to a pointer to this structure. The **ulType** member indicates what buttons to make available to the user. The **ulDefault** member indicates which button will initially be highlighted. If the **fSync** member is TRUE, the client must immediately display the dialog and wait for the user's response. If it is FALSE, the client can either return the *kNkMAIDEventResult_None* value and display the dialog at some later time or wait for the user's response.

The **lpPrompt** member will point to a null terminated string provided by the module. The pointer will be valid for the length of the user interface callback. The client must make a copy of the string in order to show the user interface asynchronously.

If more detailed information is available, the **lpDetail** member will point to a null terminated string provided by the module. The pointer will be valid for the length of the user interface callback. The client must make a copy of the string in order to show the user interface asynchronously. If more detailed information is not available, the **lpDetail** member should be set to NULL.

If the UI request is a simple message with no capabilities, the **pObject** and **data** members should be set to NULL. If capabilities are to be presented to the user, the **data** member will point to an NkMAIDArray structure allocated by the module. The array structure will contain one or more MAID capability identifiers (listed in eNkMAIDCapability). All of these capabilities must refer to the MAID object specified in **pObject**. The client will attempt to display these capabilities to the user. The array structure should be filled as follows:

```
1    ulType = kNkMAIDArrayType_Unsigned
2    ulElements = <number of capabilities to be displayed>
3    ulDimSize1 = <same as ulElements>
4    ulDimSize2 = 0
```

```
5      ulDimSize3 = 0
6      wPhysicalBytes = 4
7      wLogicalBits = 32
8      pData = <array of capability IDs allocated by the module>
```

The following table lists the likely client implementation for each type of UI Request:

Capability Type	Likely UI
Process	Button
Boolean	Check Box
Integer, Unsigned, Float, String	Edit Control
Point, Size, Rect	Custom UI
DateTime	Edit Controls or Custom UI
Callback	Undefined
Array	Radio Button Group
Range	Slider or Spin Control

5.23 Generic Data Delivery Structure

```
typedef struct tagNkMAIDDataInfo
{
    ULONG        ulType;                // one of eNkMAIDDataObjType
} NkMAIDDataInfo, FAR* LPNkMAIDDataInfo;
```

This structure is used in the NkMAIDImageInfo, NkMAIDSoundInfo, and NkMAIDFileInfo structures to indicate what type of data is being delivered to the client's data delivery callback function. If kNkMAIDDataObjType_File is combined with other value to make up the **ulType** member, it means that the accompanying data is formatted as a file and that the NkMAIDFileInfo structure should be used.

5.24 Image Data Delivery Structure

```
typedef struct tagNkMAIDImageInfo
{
    NkMAIDDataInfo    base;
    NkMAIDSize         szTotalPixels;    // total size of image to be transfered
    ULONG             ulColorSpace;      // One of eNkMAIDColorSpace
    NKMAIDRect         rData;            // coords of data, (0,0) = top left
    ULONG             ulRowBytes;        // number of bytes per row of pixels
    WORD              wBits[4];          // number of bits per plane per pixel
    WORD              wPlane;            // see below for description
    BOOL              fRemoveObject;     // TRUE if the object should be removed
} NkMAIDImageInfo, FAR* LPNkMAIDImageInfo;
```

The module sets the pDataInfo parameter of the client's data delivery callback function to a pointer to this structure to describe the image data being delivered. The **ulColorSpace** and **wBits** members apply to the image as a whole. If only one plane of a color image is being transferred, **ulColorSpace** will be the color space of the entire image all of the elements of **wBits** will be set.

If the data is being delivered in kNkMAIDColorSpace_LineArt or kNkMAIDColorSpace_Grey, the **wPlane** parameter will be ignored. If the data is being delivered in one of the color formats and is being delivered one plane at a time, the **wPlane** parameter will indicate the plane being delivered. For RGB and sRGB: R=1, G=2, B=3. For CMYK: C=1, M=2, Y=3, K=4. For Lab: L=1, A=2, B=3. For LCH: L=1, C=2, H=3. If the data is being delivered in one of the color formats and is being delivered in "chunky" format, the **wPlane** parameter will be 0.

Chunky color data will always be delivered interleaved, in the order specified by ulColorSpace (RGB, CMYK, LAB, or LCH order), LSB aligned and byte aligned. This means that 10 bit per color data will occupy two bytes per color per

pixel and the valid bits will be in the lower 10 bits of each two byte pair. The byte order is specific to the system. For the Windows environment, the low byte will be first. For the Macintosh environment, the high byte will be first.

The module can request that the data object be removed after the client is finished receiving the data by setting the **fRemoveObject** flag to TRUE. The client is not required to act on this request. If data is being delivered in more than one section, this flag should only be set to TRUE during the delivery of the final section of the data, or some data may be lost. The client may remove the data object by using the `kNkMAIDCapability_Remove` capability. If the current object is the only data object in the item object, the module may delete that item object. In this case, the module should send a `kNkMAIDEvent_RemoveChild` event to the source object.

5.25 Sound Data Delivery Structure

```
typedef struct tagNkMAIDSoundInfo
{
    NkMAIDDataInfo    base;
    ULONG             ulTotalSamples;    // number of full samples to be transferred
    BOOL              fStereo;           // TRUE if stereo, FALSE if mono
    ULONG             ulStart;           // index of starting sample of data
    ULONG             ulLength;          // number of samples of data
    WORD              wBits;             // number of bits per channel
    WORD              wChannel;          // 0 = mono or L+R; 1,2 = left, right
    BOOL              fRemoveObject;     // TRUE if the object should be removed
} NkMAIDSoundInfo, FAR* LPNkMAIDSoundInfo;
```

The module sets the `pDataInfo` parameter of the client's data delivery callback function to a pointer to this structure to describe the sound data being delivered. The **fStereo** member applies to the sound as a whole. If only one channel of a stereo sound is being transferred, it will be TRUE.

Stereo data will always be delivered interleaved, in LR order, LSB aligned and byte aligned. This means that 10 bit per channel data will occupy two bytes per channel per sample and the valid bits will be in the lower 10 bits of each two byte pair. The byte order is specific to the system. For the Windows environment, the low byte will be first. For the Macintosh environment, the high byte will be first.

The module can request that the data object be removed after the client is finished receiving the data by setting the **fRemoveObject** flag to TRUE. The client is not required to act on this request. If data is being delivered in more than one section, this flag should only be set to TRUE during the delivery of the final section of the data, or some data may be lost. The client may remove the data object by using the `kNkMAIDCapability_Remove` capability. If the current object is the only data object in the item object, the module may delete that item object. In this case, the module should send a `kNkMAIDEvent_RemoveChild` event to the source object.

5.26 Enumeration Structure

```
typedef struct tagNkMAIDEnum
{
    ULONG             ulType;            // one of eNkMAIDArrayType
    ULONG             ulElements;        // total number of elements
    ULONG             ulValue;           // current index (zero-based)
    ULONG             ulDefault;         // default index (zero-based)
    WORD              wPhysicalBytes;    // bytes per element
    LPVOID            pData;             // allocated by the client
} NkMAIDEnum, FAR* LPNkMAIDEnum;
```

The `NkMAIDEnum` structure allows an enumeration to be transferred through the MAID interface. The client will always allocate the memory. It is the responsibility of the receiver of the data to interpret the data properly. The size of **pData** in bytes should always be **ulElements** times **wPhysicalBytes**.

This structure is used to implement a capability that is a choice of options. The current index will be in **ulValue** and the default index will be in **ulDefault**. If **ulType** is `kNkMAIDArrayType_String` or `kNkMAIDArrayType_PackedString`, the strings are the text representations to be presented to the user.

The value of a choice capability can be set in two ways. The client can send a pointer to an `NkMAIDEnum` structure with a new index in **ulValue** or it can send an unsigned integer which will be interpreted as the index.

When **ulType** is `kNkMAIDArrayType_PackedString`, **pData** will point to a packed list of null terminated strings, **ulElements** will be the total length in bytes of the data including the terminating null bytes and **wPhysicalBytes** will be one.

5.27 File Data Delivery Structure

```
typedef struct tagNkMAIDFileInfo
{
    NkMAIDDataInfo    base;
    ULONG             ulFileType;           // One of eNkMAIDFileDataTypes
    ULONG             ulTotalLength;        // total number of bytes to be transferred
    ULONG             ulStart;              // index of starting byte (0-based)
    ULONG             ulLength;             // number of bytes in this delivery
    BOOL              fDiskFile;           // TRUE if the file is delivered on disk
    BOOL              fRemoveObject;       // TRUE if the object should be removed
} NkMAIDFileInfo, FAR* LPNkMAIDFileInfo;
```

The module sets the `pDataInfo` parameter in the client's data delivery callback function to a pointer to this structure to describe the data being delivered. The `ulFileType` and `ulTotalLength` members apply to the file data as a whole (the type and size of the actual file). The `ulStart` member will contain the offset in the file of the data being delivered. The `ulLength` member will contain the length of the data being delivered. If the data is delivered in more than one call, the pieces should be delivered in order from beginning to end.

If the file being delivered is on disk, the `fDiskFile` member should be set to `TRUE` and the `pData` member of the client's data delivery callback will be set to a pointer to an `NkMAIDString` structure. This structure will contain the full path and name of the disk file. When delivering a file on disk, the `ulStart` member should be set to 0; the `ulLength` and `ulTotalLength` members should be set to the total length of the file, if it is known by the module. If the length of the file is not known by the module, these members may be set to 0. The module may not deliver data in a disk file unless the client has set the delivery location with the `kNkMAIDCapability_AcceptDiskAcquisition` capability first.

The module can request that the data object be removed after the client is finished receiving the data by setting the **fRemoveObject** flag to `TRUE`. The client is not required to act on this request. If data is being delivered in more than one section, this flag should only be set to `TRUE` during the delivery of the final section of the data, or some data may be lost. The client may remove the data object by using the `kNkMAIDCapability_Remove` capability. If the current object is the only data object in the item object, the module may delete that item object. In this case, the module should send a `kNkMAIDEvent_RemoveChild` event to the source object.

6 Result Codes

One of these values will be returned from the entry point function and sent to the completion callback function.

6.1 kNkMAIDResult_NotSupported

The module will return this value if the client attempted to perform an operation on a capability that does not exist for the specified object or if the client attempted to perform an operation that is not supported for the capability.

6.2 kNkMAIDResult_UnexpectedDataType

The module will return this value if the client sent the **ulDataType** parameter to the entry point function to a type that is incorrect for the command and/or capability.

6.3 kNkMAIDResult_ValueOutOfBounds

The module will return this value if the client attempts to set a capability to a value outside the allowed range for that capability.

6.4 kNkMAIDResult_BufferSize

The module will return this value in only two cases. When the client sends a *kNkMAIDCommand_GetCapInfo* command and the count does not match the number of capabilities. When the client sends a *kNkMAIDCommand_CapGet* for an array capability and the size specified in the array structure does not match the size of the data for that capability.

6.5 kNkMAIDResult_Aborted

The module will return this value for an asynchronous command if the client sends the *kNkMAIDCommand_Abort*, *kNkMAIDCommand_AbortToMark* or *kNkMAIDCommand_Close* command for the asynchronous command's object

6.6 kNkMAIDResult_NoMedia

The module will return this value if the client attempts to start an acquisition, autofocus, eject or some other process capability that requires some media in the device.

6.7 kNkMAIDResult_NoEventProc

The module will return this value if the client sends the *kNkMAIDCommand_EnumChildren* command without first setting the *kNkMAIDCapability_EventProc* capability to a value other than NULL.

6.8 kNkMAIDResult_ZombieObject

The module will return this value if the client attempts to send a command that cannot be completed because the object is no longer alive.

6.9 kNkMAIDResult_NoError

The module will return this value if the command completed successfully.

6.10 kNkMAIDResult_Pending

The module will return this value if the client specified a completion callback function for a command and the module wants to return control to the client before the command is complete.

6.11 kNkMAIDResult_OrphanedChildren

The module will return this value if the client closes an object while the client still has children of that object open.

6.12 kNkMAIDResult_NoDataProc

The module will return this value if the client starts an acquisition with *kNkMAIDCapability_Acquire* and has not specified a DataProc for the object.

6.13 kNkMAIDResult_OutOfMemory

The module will return this value if some operation cannot be completed because of a low-memory condition.

6.14 kNkMAIDResult_UnexpectedError

The module will return this value if some operation cannot be completed because of an unexpected error.

6.15 kNkMAIDResult_HardwareError

The module will return this value if some operation cannot be completed because of a hardware error.

6.16 kNkMAIDResult_MissingComponent

The module will return this value if some operation cannot be completed because of a failure to find, open, or access a required file.

7 Events

Events are optional for the client, but not for the module. All of the conditions that the client would be notified of through the event callback function can be deduced by polling various elements.

7.1 kNkMAIDEvent_AddChild

The client can deduce this event by polling the *kNkMAIDCapability_Children* capability.

The module will send this event to the parent module, source or item object when it detects the addition of a new child source, item or data type object, respectively. If the event is sent to a module or source, the **data** parameter will be the ID of the new child. If the event is sent to an item, the **data** parameter will be one of eNkMAIDDataType.

The client can send the *kNkMAIDCommand_EnumChildren* command to request that the module enumerate all of the children of an object. The module will send a *kNkMAIDEvent_AddChild* to the object for each of the children. If there is no event callback function, the module will return the *kNkMAIDResult_NoEventProc* for the command.

7.2 kNkMAIDEvent_RemoveChild

The client can deduce this event by polling the *kNkMAIDCapability_Children* capability.

The module will send this event to the parent module, source or item object when it detects the removal of a child source, item or data type object, respectively. If the event is sent to a module or source, the **data** parameter will be the ID of the new child. If the event is sent to an item, the **data** parameter will be one of eNkMAIDDataType.

Before sending this event to the parent object, the module will first abort any asynchronous commands currently being processed for the child object and then set the child object's *kNkMAIDCapability_IsAlive* capability to FALSE.

7.3 kNkMAIDEvent_WarmingUp

The client can deduce this event by examining the *kNkMAIDCapability_WarmedUp* capability.

The module will send this event to a source object when the device enters a state where it cannot guarantee the best quality, for instance, when a light source is first turned on.

7.4 kNkMAIDEvent_WarmedUp

The client can deduce this event by examining the *kNkMAIDCapability_WarmedUp* capability.

The module will send this event to a source object when the device leaves a state where it cannot guarantee the best quality, for instance, when a light source is first turned on.

7.5 kNkMAIDEvent_CapChange

The client can deduce this event by examining the value and number of capabilities.

The module will send this event to the module, source, item, or data object when the number of capabilities or the values of any of the existing capabilities has changed. Normally, if the client sets the value of a single capability by using the *kNkMAIDCommand_CapSet* command, this event is not required. However, if capabilities other than the one specified by the client are affected by the *kNkMAIDCommand_CapSet* command, this event should be sent.

If this event is sent to indicate a single capability change, the **data** parameter will be the ID of the capability which has changed. If this event is sent to indicate multiple capability changes or a change to the number of capabilities available, the **data** parameter will be NULL. If the values of several capabilities have changed, the module has the option of sending one CapChange event for each capability, or a single CapChange event with NULL data.

7.6 kNkMAIDEvent_OrphanedChildren

The client can deduce this event by examining the *kNkMAIDCapability_IsAlive* capability of the children objects.

The module will send this event to notify an object that is being closed that it has children objects that are still open.

7.7 kNkMAIDEvent_CapChangeValueOnly

The client can deduce this event by examining the value and number of capabilities.

The module will send this event to the module, source, item, or data object when the current value of a capability has changed. This event implies that characteristics other than the current value have not changed (e.g. number of array elements, enumeration data, visibility, available operations, etc.). If any other characteristics have changed, the *kNkMAIDEvent_CapChange* event should be sent instead. Normally, if the client sets the value of a single capability by using the *kNkMAIDCommand_CapSet* command, this event is not required. However, if capabilities other than the one specified by the client are affected by the *kNkMAIDCommand_CapSet* command, this event should be sent.

If this event is sent to indicate a single capability change, the **data** parameter will be the ID of the capability which has changed. If this event is sent to indicate changes to a number of capabilities, the **data** parameter will be NULL. If the values of several capabilities have changed, the module has the option of sending one CapChangeValueOnly event for each capability, or a single CapChangeValueOnly event with NULL data.

8 Commands

For each of the commands documented here, there is an explanation of what the parameters to the MAID entry point will be.

Any command may take a significant amount of time to process. It is at the module's discretion as to whether to complete the command synchronously during one call to the module or return immediately and process the command asynchronously in another thread or during *kNkMAIDCommand_Async* commands. When the command is complete, the completion function supplied with the command will be called whether it was processed synchronously or asynchronously. The client can require that the command be processed synchronously by not specifying a completion function pointer. Asynchronous commands that are issued while the client is processing a callback (e.g. completion callback or event notification callback) may or may not be completed (at the module's discretion) until the client exits that callback.

The client may send several related asynchronous commands without waiting for previous ones to complete. The last one of the series will be the *kNkMAIDCommand_Mark* command. If one of the queued commands fails, the client can send the *kNkMAIDCommand_AbortToMark* command to abort the other commands up to and including the *kNkMAIDCommand_Mark* command. The module will call the completion function for each command with the **nResult** parameter set to *kNkMAIDResult_Aborted*.

An issue arises when the module processes some commands synchronously and others asynchronously. If a synchronous command is preceded by a command that is being processed asynchronously, the module must determine whether it is safe to process the new command immediately or not. If the module decides to delay processing the new command, the module must then decide whether to wait until after the previous command is complete and then process the new command synchronously or queue the new command to be processed asynchronously after the previous command.

For array and enumeration capabilities, two commands must be sent to get the data. To know how much memory to allocate, the client must first send the *kNkMAIDCommand_CapGet* command. The module will set all the members of the *NkMAIDArray* or *NkMAIDEnum* structure. Once the client allocates the memory and sets the **pData** member of the *NkMAIDArray* or *NkMAIDEnum* structure, that structure will be sent for the *kNkMAIDCommand_CapGetArray* command. If the size of the data changes between these two calls, the module will not store any data in the **pData** member and return *kNkMAIDResult_BufferSize*. The client should start the process over by sending another *kNkMAIDCommand_CapGet* command.

8.1 kNkMAIDCommand_Async

This command will process asynchronous commands for the specified object in a single threaded module.

pObject	May be NULL or refer to a module, source, item or data object
ulParam	Ignored
ulDataType	Ignored
data	Ignored

Multithreaded or synchronous modules can simply return *kNkMAIDResult_NoError* for this command.

If the module has the *kNkMAIDCapability_AsyncRate* capability, the client will send this command at the specified rate during idle periods.

The module will return *kNkMAIDResult_Pending* to indicate that it is processing commands and it wishes to receive this command as soon as possible and not at the rate specified by the *kNkMAIDCapability_AsyncRate* capability

8.2 kNkMAIDCommand_Open

This command will open a child of the specified object.

pObject	May be NULL or refer to a module, source or item
ulParam	May be NULL, the ID of the source or item or the type of the data object to be opened
ulDataType	Must be <i>kNkMAIDDataType_ObjectPtr</i>
data	Must be a pointer to a NkMAIDObject structure

This command will be the first command sent to a module after it is loaded. When the **pObject** parameter is NULL, the object opened will be a module object and the module will initialize itself if it has not already. When the **pObject** parameter refers to a module, source or item, the object opened will be a source, item or data object, respectively.

The client will set the **refClient** member of the NkMAIDObject structure passed in the **data** parameter before calling the module. The module will allocate any internal structures to maintain a state and store a pointer, handle, ID or other identifier in the **refModule** member of the NkMAIDObject structure. The module will set all of the NkMAIDObject structure members except **refClient**. While the object is open, the client will not change the value of the **refClient** member and the module will not change the value of the **refModule** member. No two objects can have the same value for **refClient** or **refModule**.

This command may be used more than once by the same client to open the same module, source, item or data object. The module will maintain a separate internal structure and state for each invocation.

If the command completes successfully, the client must close the object before releasing the module.

8.3 kNkMAIDCommand_Close

This command will close the connection to the specified module, source, item or data object.

pObject	May refer to a module, source, item or data object
ulParam	Ignored
ulDataType	Ignored
data	Ignored

The module will abort any commands that are processing asynchronously for the object. The module will set the **refModule** member of the NkMAIDObject structure to NULL. The client may not use the structure again without reopening the object.

If the client has not closed all of the child objects of the object it is closing, the module will send the *kNkMAIDEvent_OrphanedChildren* event. If there is no event callback function or the event callback function does not close all of the child objects, the module will return *kNkMAIDResult_OrphanedChildren*.

8.4 kNkMAIDCommand_GetCapCount

This command will get the number of capabilities available for the specified module, source, item or data object.

pObject	May refer to a module, source, item or data object
ulParam	Ignored
ulDataType	Must be <i>kNkMAIDDataType_UnsignedPtr</i>
data	Must be a pointer to a 32 bit unsigned integer.

8.5 kNkMAIDCommand_GetCapInfo

This command will get information about all of the capabilities available for the specified module, source, item or data object.

pObject	May refer to a module, source, item or data object
ulParam	The number of NkMAIDCapInfo structures that can be stored
ulDataType	Must be <i>kNkMAIDDataType_CapInfoPtr</i>
data	Must be a pointer to an array of NkMAIDCapInfo structures

The size of the array must coincide with the **ulParam** parameter. The module will return *kNkMAIDResult_BufferSize* if it does not.

8.6 kNkMAIDCommand_CapStart

This command will start the specified capability for the specified module, source, item or data object.

pObject	May refer to a module, source, item or data object
ulParam	The ID of the capability to be started
ulDataType	Ignored
data	Ignored

Within the capability's NkMAIDCapInfo structure, the **ulType** member must be *kNkMAIDCapType_Process* and the **ulOperations** member must be have the *kNkMAIDCapOperation_Start* bit set. If the capability does not support this command, the module will return *kNkMAIDResult_NotSupported*.

8.7 kNkMAIDCommand_CapSet

This command will set the value of the specified capability for the specified module, source, item or data object.

pObject	May refer to a module, source, item or data object
ulParam	The ID of the capability to be set
ulDataType	One of eNkMAIDDataType
data	Value or pointer

Within the capability's NkMAIDCapInfo structure, the **ulOperations** member must be have the *kNkMAIDCapOperation_Set* bit set. The values that are permitted for **ulDataType** depends on the **ulType** member of the capability's NkMAIDCapInfo structure according to this table:

<u>ulType value</u>	<u>ulDataType value</u>
<i>kNkMAIDCapType_Boolean</i>	<i>kNkMAIDDataType_Boolean, kNkMAIDDataType_BooleanPtr</i>
<i>kNkMAIDCapType_Integer</i>	<i>kNkMAIDDataType_Integer, kNkMAIDDataType_IntegerPtr</i>
<i>kNkMAIDCapType_Unsigned</i>	<i>kNkMAIDDataType_Unsigned, kNkMAIDDataType_UnsignedPtr</i>
<i>kNkMAIDCapType_Float</i>	<i>kNkMAIDDataType_FloatPtr</i>
<i>kNkMAIDCapType_Point</i>	<i>kNkMAIDDataType_PointPtr</i>
<i>kNkMAIDCapType_Size</i>	<i>kNkMAIDDataType_SizePtr</i>
<i>kNkMAIDCapType_Rect</i>	<i>kNkMAIDDataType_RectPtr</i>
<i>kNkMAIDCapType_String</i>	<i>kNkMAIDDataType_StringPtr</i>
<i>kNkMAIDCapType_DateTime</i>	<i>kNkMAIDDataType_DateTimePtr</i>
<i>kNkMAIDCapType_Callback</i>	<i>kNkMAIDDataType_CallbackPtr, kNkMAIDDataType_Null</i>
<i>kNkMAIDCapType_Array</i>	<i>kNkMAIDDataType_ArrayPtr</i>
<i>kNkMAIDCapType_Enum</i>	<i>kNkMAIDDataType_EnumPtr, kNkMAIDDataType_Unsigned</i>
<i>kNkMAIDCapType_Range</i>	<i>kNkMAIDDataType_RangePtr, kNkMAIDDataType_Unsigned</i>
<i>kNkMAIDCapType_Generic</i>	<i>kNkMAIDDataType_GenericPtrRangePtr</i>

This command is not permitted for the *kNkMAIDCapType_Process* type.

If the data type does not match this table, the module will return *kNkMAIDResult_UnexpectedDataType*. If the capability does not support this command, the module will return *kNkMAIDResult_NotSupported*.

8.8 kNkMAIDCommand_CapGet

This command will get the value of the specified capability for the specified module, source, item or data object.

pObject	May refer to a module, source, item or data object
ulParam	The ID of the capability to be retrieved
ulDataType	One of eNkMAIDDataType
data	Pointer

Within the capability's NkMAIDCapInfo structure, the **ulOperations** member must be have the *kNkMAIDCapOperation_Get* bit set. The values that are permitted for **ulDataType** depends on the **ulType** member of the capability's NkMAIDCapInfo structure according to this table:

<u>ulType value</u>	<u>ulDataType value</u>
<i>kNkMAIDCapType_Boolean</i>	<i>kNkMAIDDataType_BooleanPtr</i>
<i>kNkMAIDCapType_Integer</i>	<i>kNkMAIDDataType_IntegerPtr</i>
<i>kNkMAIDCapType_Unsigned</i>	<i>kNkMAIDDataType_UnsignedPtr</i>
<i>kNkMAIDCapType_Float</i>	<i>kNkMAIDDataType_FloatPtr</i>
<i>kNkMAIDCapType_Point</i>	<i>kNkMAIDDataType_PointPtr</i>
<i>kNkMAIDCapType_Size</i>	<i>kNkMAIDDataType_SizePtr</i>
<i>kNkMAIDCapType_Rect</i>	<i>kNkMAIDDataType_RectPtr</i>
<i>kNkMAIDCapType_String</i>	<i>kNkMAIDDataType_StringPtr</i>
<i>kNkMAIDCapType_DateTime</i>	<i>kNkMAIDDataType_DateTimePtr</i>
<i>kNkMAIDCapType_Callback</i>	<i>kNkMAIDDataType_CallbackPtr</i>
<i>kNkMAIDCapType_Array</i>	<i>kNkMAIDDataType_ArrayPtr</i>
<i>kNkMAIDCapType_Enum</i>	<i>kNkMAIDDataType_EnumPtr</i>
<i>kNkMAIDCapType_Range</i>	<i>kNkMAIDDataType_RangePtr</i>
<i>kNkMAIDCapType_Generic</i>	<i>kNkMAIDDataType_GenericPtr</i>

This command is not permitted for the *kNkMAIDCapType_Process* type.

If the data type does not match this table, the module will return *kNkMAIDResult_UnexpectedDataType*. If the capability does not support this command, the module will return *kNkMAIDResult_NotSupported*.

8.9 kNkMAIDCommand_CapGetDefault

This command will get the default value of the specified capability for the specified module, source, item or data object.

pObject	May refer to a module, source, item or data object
ulParam	The ID of the capability to be retrieved
ulDataType	One of eNkMAIDDataType
data	Pointer

Within the capability's NkMAIDCapInfo structure, the **ulOperations** member must be have the *kNkMAIDCapOperation_GetDefault* bit set. The values that are permitted for **ulDataType** depends on the **ulType** member of the capability's NkMAIDCapInfo structure according to this table:

ulType value	ulDataType value
<i>kNkMAIDCapType_Boolean</i>	<i>kNkMAIDDataType_BooleanPtr</i>
<i>kNkMAIDCapType_Integer</i>	<i>kNkMAIDDataType_IntegerPtr</i>
<i>kNkMAIDCapType_Unsigned</i>	<i>kNkMAIDDataType_UnsignedPtr</i>
<i>kNkMAIDCapType_Float</i>	<i>kNkMAIDDataType_FloatPtr</i>
<i>kNkMAIDCapType_Point</i>	<i>kNkMAIDDataType_PointPtr</i>
<i>kNkMAIDCapType_Size</i>	<i>kNkMAIDDataType_SizePtr</i>
<i>kNkMAIDCapType_Rect</i>	<i>kNkMAIDDataType_RectPtr</i>
<i>kNkMAIDCapType_Generic</i>	<i>kNkMAIDDataType_GenericPtr</i>

This command is not permitted for the *kNkMAIDCapType_Process*, *kNkMAIDCapType_String*, *kNkMAIDCapType_DateTime*, *kNkMAIDCapType_Callback*, *kNkMAIDCapType_Array*, *kNkMAIDCapType_Enum*, and *kNkMAIDCapType_Range* types.

If the data type does not match this table, the module will return *kNkMAIDResult_UnexpectedDataType*. If the capability does not support this command, the module will return *kNkMAIDResult_NotSupported*.

8.10 kNkMAIDCommand_CapGetArray

This command will get the data associated with the specified array capability for the specified module, source, item or data object.

pObject	May refer to a module, source, item or data object
ulParam	The ID of the array capability for which to get data
ulDataType	Must be <i>kNkMAIDDataType_ArrayPtr</i> , <i>kNkMAIDDataType_EnumPtr</i>
data	Must be a pointer to an NkMAIDArray or NkMAIDEnum structure

Within the capability's NkMAIDCapInfo structure, the **ulType** member must be *kNkMAIDCapType_Array* or *kNkMAIDCapType_Enum* and the **ulOperations** member must have the *kNkMAIDCapOperation_GetArray* bit set. All of the members of the NkMAIDArray or NkMAIDEnum structure will not be changed, the module will only store data at the address pointed to by the **pData** member. If the capability does not support this command, the module will return *kNkMAIDResult_NotSupported*. The module will return *kNkMAIDResult_BufferSize* if the members of the NkMAIDArray or NkMAIDEnum structure do not match what the module wants to store.

8.11 kNkMAIDCommand_Mark

This command will insert a mark in the queue for the specified module, source, item or data object.

pObject	May refer to a module, source, item or data object
ulParam	Ignored
ulDataType	Ignored
data	Ignored

This command is to better support asynchronous command processing by the module. The module does not need to perform any operations, but this command must not complete until all of the asynchronous commands before it for the specified object are complete. A completion function might not be supplied for this command.

8.12 kNkMAIDCommand_AbortToMark

This command will abort asynchronous commands in the queue for the specified module, source, item or data object up to and including the next *kNkMAIDCommand_Mark* command.

pObject	May refer to a module, source, item or data object
ulParam	Ignored
ulDataType	Ignored
data	Ignored

This command is to better support asynchronous command processing by the module.

The client may send several related asynchronous commands without waiting for previous ones to complete. The last one of the series will be the *kNkMAIDCommand_Mark* command. If one of the queued commands fails, the client can send the *kNkMAIDCommand_AbortToMark* command to abort the other commands up to and including the *kNkMAIDCommand_Mark* command. The module will call the completion function for each command with the **nResult** parameter set to *kNkMAIDResult_Aborted*.

If there is no *kNkMAIDCommand_Mark* command, all asynchronous commands will be aborted.

Only commands sent to the specified object will be aborted.

8.13 kNkMAIDCommand_Abort

This command will abort the asynchronous command currently being processed.

pObject	May refer to a module, source, item or data object
ulParam	Ignored
ulDataType	Ignored
data	Ignored

This command is to better support asynchronous command processing by the module. The module will call the completion function for the command with *kNkMAIDResult_Aborted*.

Only commands sent to the specified object will be aborted.

8.14 kNkMAIDCommand_EnumChildren

The module will send *kNkMAIDEvent_AddChild* events to the object for all of it's children.

pObject	May refer to a module, source or item
ulParam	Ignored
ulDataType	Ignored
data	Ignored

If the client did not set the *kNkMAIDCapability_EventProc* capability to a value other than NULL before sending this command, the module will return *kNkMAIDResult_NoEventProc*.

8.15 kNkMAIDCommand_GetParent

The module will get information about an object's parent.

pObject	May refer to a source, item or data object
ulParam	Ignored
ulDataType	Must be <i>kNkMAIDDataType_ObjectPtr</i>
data	Must be a pointer to a NkMAIDObject structure

The module will set the members of the NkMAIDObject structure to match the values of the object's parent's NkMAIDObject structure. The client may use the structure in subsequent calls to the module. It is, however, the responsibility of the client to make sure any persistent data it has for the parent object remains valid.

8.16 kNkMAIDCommand_ResetToDefault

The module will reset the selected object to its default values.

pObject	May refer to a module, source, item or data object
ulParam	Ignored
ulDataType	Ignored
data	Ignored

The module will reset all capabilities of the selected object to their default values. If the selected object has open children or data objects, those objects should also have their capabilities reset to their default values.

9 Capabilities

For each of the capabilities documented here, there is an explanation of what the members of the `NkMAIDCapInfo` structure will be.

Most of the capabilities listed here will be handled explicitly by the client. The remaining listed capabilities and the vendor supplied capabilities will be handled in a generic manner. The client will use the **ulVisibility** and **szDescription** members to describe them to the user.

The module has the ability to specify that a set of capabilities should be treated as a group. For each capability that is to be included in a group, the **ulVisibility** member must contain the `kNkMAIDCapVisibility_GroupMember` value. In order to group those capabilities, a new "group" capability must be created. The **ulVisibility** member must contain the `kNkMAIDCapVisibility_Group` value. This new capability will be a `kNkMAIDCapType_Array` type capability, which will contain an array of ID's of other capabilities. When the client reads this capability with the `kNkMAIDCommand_CapGet` command, the module will fill the `NkMAIDArray` structure as follows:

```
1      ulType = kNkMAIDArrayType_Unsigned
2      ulElements = <number of member-capabilities in this group>
3      ulDimSize1 = <same as ulElements>
4      ulDimSize2 = 0
5      ulDimSize3 = 0
6      wPhysicalBytes = 4
7      wLogicalBits = 32
8      pData = NULL
```

After allocating enough memory to hold the data, the client will call the module with the `kNkMAIDCommand_CapGetArray` command. Upon return, the `pData` member should contain an array of ULONG values, each one of which is the ID of another capability.

9.1 kNkMAIDCapability_AsyncRate

The module uses this capability to suggest the frequency that the client should send `kNkMAIDCommand_Async` commands during idle periods. It is expressed as the number of milliseconds (1/1000 s) between `kNkMAIDCommand_Async` commands.

Object types Module only
ulType `kNkMAIDCapType_Unsigned`
ulOperations `kNkMAIDCapOperation_Get`

This capability should not be provided if a module does not require periodic `kNkMAIDCommand_Async` commands, as may be the case with a multithreaded module.

The frequency is merely a suggestion from the module to the client. The client may not be able to send the commands as fast as the module would like them.

See the description of the `kNkMAIDCommand_Async` command for more information.

9.2 kNkMAIDCapability_ProgressProc

The module will call this callback during lengthy processes.

Object types Module, source, item or data object
ulType *kNkMAIDCapType_Callback*
ulOperations *kNkMAIDCapOperation_Get, kNkMAIDCapOperation_Set*

If a command issued for the object will take a significant amount of time, the module will call this callback so the client can provide the user with a progress display. How often, or whether to call this callback at all, is at the module's discretion.

The initial value will be NULL. The client can indicate it does not want to be given progress information by setting this capability with the **ulDataType** parameter to the MAID entry point set to *kNkMAIDDataType_Null*.

See the description of the MAIDProgress callback function for more information.

9.3 kNkMAIDCapability_EventProc

The module will call this callback to notify the client of events.

Object types Module, source, item or data object
ulType *kNkMAIDCapType_Callback*
ulOperations *kNkMAIDCapOperation_Get, kNkMAIDCapOperation_Set*

The initial value will be NULL. The client can indicate it does not want to be given event notification by setting this capability with the **ulDataType** parameter to the MAID entry point set to *kNkMAIDDataType_Null*.

See the description of the MAIDEvent callback function for more information.

9.4 kNkMAIDCapability_DataProc

The module will call this callback to deliver data to the client.

Object types Data object only
ulType *kNkMAIDCapType_Callback*
ulOperations *kNkMAIDCapOperation_Get, kNkMAIDCapOperation_Set*

The module is required to provide this capability for data objects.

The initial value will be NULL. The client must set this capability before starting an acquire. Once the data exchange is complete, the client can set this capability with the **ulDataType** parameter to the MAID entry point set to *kNkMAIDDataType_Null*.

See the description of the MAIDData callback function for more information.

9.5 kNkMAIDCapability_UIRequestProc

The module will call this callback to request that some user interface be shown.

Object types Module only
ulType *kNkMAIDCapType_Callback*
ulOperations *kNkMAIDCapOperation_Get, kNkMAIDCapOperation_Set*

The module is required to provide this capability for module objects. The client must set this capability just after the module is opened. If it does not, the module might not be able to notify the user or ask the user a question.

The initial value will be NULL.

See the description of the MAIDUIRequest callback function for more information.

9.6 kNkMAIDCapability_IsAlive

This is the objects validity state.

Object types Module, source, item or data object
ulType *kNkMAIDCapType_Boolean*
ulOperations *kNkMAIDCapOperation_Get*

The module is required to provide this capability for all objects.

The value of this capability is usually TRUE. It is FALSE if the object is removed by the module or the object's parent is closed by the client.

9.7 kNkMAIDCapability_Children

This is the list of child source or item IDs.

Object types Module or source
ulType *kNkMAIDCapType_Enum*
ulOperations *kNkMAIDCapOperation_Get, kNkMAIDCapOperation_GetArray*

The module is required to provide this capability for module and source objects.

Within the NkMAIDArray structure, **ulType** will be *kNkMAIDArrayType_Unsigned* and **wPhysicalBytes** will be four .

9.8 kNkMAIDCapability_State

The client can use this capability save the state of the object for later retrieval.

Object types Module, source, item or data object
ulType *kNkMAIDCapType_Array*
ulOperations *kNkMAIDCapOperation_GetArray, kNkMAIDCapOperation_Get, kNkMAIDCapOperation_Set*

The data within the array is entirely module dependent and will not be interpreted by the client. The data will be saved and restored by the client verbatim.

Within the NkMAIDArray structure, **ulType** will be *kNkMAIDArrayType_Unsigned*, **wPhysicalBytes** will be one and **wLogicalBits** will be eight.

9.9 kNkMAIDCapability_Name

This is the name of the object.

Object types Module, source, item or data object
ulType *kNkMAIDCapType_String*
ulOperations *kNkMAIDCapOperation_Get*

Unlike the *kNkMAIDCapability_Description* capability, this capability cannot be set. The module should use that capability if it can store a descriptive name and it wishes to allow the user to edit that name.

9.10 kNkMAIDCapability_Description

This is the description of the object.

Object types Module, source, item or data object
ulType *kNkMAIDCapType_String* or *kNkMAIDCapType_Array*
ulOperations *kNkMAIDCapOperation_Get, kNkMAIDCapOperation_GetArray* for an array type, possibly *kNkMAIDCapOperation_Set*

The module will provide this capability if it can describe the object in more detail than the *kNkMAIDCapability_Name* capability. This will allow the user to better identify the object described.

The module may implement this capability as an array. The **ulType** member of the `NkMAIDArray` structure will be *kNkMAIDArrayType_String* or *kNkMAIDArrayType_PackedString*.

9.11 kNkMAIDCapability_Interface

This is the description of the physical interface being used to communicate with the source.

Object types Source only
ulType *kNkMAIDCapType_String*
ulOperations *kNkMAIDCapOperation_Get*

This will allow the user to better identify the source.

9.12 kNkMAIDCapability_DataTypes

This is the data types available from an item or the data types that a source can produce.

Object types Item or source
ulType *kNkMAIDCapType_Unsigned*
ulOperations *kNkMAIDCapOperation_Get*

The value will be a bitwise combination of the `eNkMAIDDataObjType` values. The value `kNkMAIDDataObjType_File` should not be used in connection with this capability. This value is to be used only for data delivery.

9.13 kNkMAIDCapability_DateTime

This is the date and time of the item's capture.

Object types Item only
ulType *kNkMAIDCapType_DateTime*
ulOperations *kNkMAIDCapOperation_Get*

This capability will only be provided by modules for devices with storage capabilities. A scanner module will not provide it.

9.14 kNkMAIDCapability_StoredBytes

This is the size of the object in bytes as it is stored in the device.

Object types Item or data object
ulType *kNkMAIDCapType_Unsigned*
ulOperations *kNkMAIDCapOperation_Get*

This capability will only be provided by modules for devices with storage capabilities.

9.15 kNkMAIDCapability_Eject

This will eject the media from the source device.

Object types Source or item
ulType *kNkMAIDCapType_Process*
ulOperations *kNkMAIDCapOperation_Start*

If this capability is started for a source, all media will be ejected. If it is started for an item, only the media for that item will be ejected.

9.16 kNkMAIDCapability_Feed

This will feed media into the source device.

Object types Source only
ulType *kNkMAIDCapType_Process*
ulOperations *kNkMAIDCapOperation_Start*

If there is no media to be fed, the module will return *kNkMAIDResult_NoMedia*.

9.17 kNkMAIDCapability_Capture

This will capture another item for the source device.

Object types Source only
ulType *kNkMAIDCapType_Process*
ulOperations *kNkMAIDCapOperation_Start*

Upon successful completion of this process, the source will have an addition child item. The source should enumerate it's items again.

9.18 kNkMAIDCapability_Mode

This is the acquire mode for the data object.

Object types Data object only
ulType *kNkMAIDCapType_Enum*
ulOperations *kNkMAIDCapOperation_Get, kNkMAIDCapOperation_GetArray, kNkMAIDCapOperation_Set*

It is up to the module to decide what modes are available and what they mean. The user will make the choice from the string array. The **ulType** member of the *NkMAIDEnum* structure will be *kNkMAIDArrayType_String* or *kNkMAIDArrayType_PackedString*.

9.19 kNkMAIDCapability_Acquire

This will start the acquire.

Object types Data object only
ulType *kNkMAIDCapType_Process*
ulOperations *kNkMAIDCapOperation_Start*

The module will begin calling the data object's data delivery callback with data. The module may also call the data object's progress callback if the acquire will take a significant amount of time.

9.20 kNkMAIDCapability_Start

The starting position for the acquire in seconds.

Object types Sound or Video
ulType *kNkMAIDCapType_Float*
ulOperations *kNkMAIDCapOperation_Get, kNkMAIDCapOperation_Set*

This is the offset from the beginning of the sound or video object. This capability will only be provided by modules for devices with storage capabilities.

9.21 kNkMAIDCapability_Length

The length available or the length to be acquired in seconds.

Object types Sound or Video
ulType *kNkMAIDCapType_Float*
ulOperations *kNkMAIDCapOperation_Get*, possibly *kNkMAIDCapOperation_Set*, possibly *kNkMAIDCapOperation_GetDefault*

The default value will be the total length available from a module for a device with storage capabilities.

9.22 kNkMAIDCapability_SampleRate

The number of samples per second to acquire.

Object types Sound or video
ulType *kNkMAIDCapType_Enum* or *kNkMAIDCapType_Range*
ulOperations *kNkMAIDCapOperation_Get*, *kNkMAIDCapOperation_GetArray*, possibly *kNkMAIDCapOperation_Set*

Within the NkMAIDEnum structure, the **ulType** member will be *kNkMAIDArrayType_Float*.

9.23 kNkMAIDCapability_Stereo

This will select the type as either mono or stereo.

Object types Sound or video
ulType *kNkMAIDCapType_Boolean*
ulOperations *kNkMAIDCapOperation_Get*; *kNkMAIDCapOperation_Set*

If the module does not provide this capability, the client will assume that the device is only capable of mono acquires.

9.24 kNkMAIDCapability_Samples

The number of samples that will be acquired with consideration given to the current state of the data object.

Object types Sound or video
ulType *kNkMAIDCapType_Unsigned*
ulOperations *kNkMAIDCapOperation_Get*

9.25 kNkMAIDCapability_Filter

This will select the filter for the light source of the device.

Object types Image or thumbnail
ulType *kNkMAIDCapType_Enum*
ulOperations *kNkMAIDCapOperation_Get*, *kNkMAIDCapOperation_GetArray*, *kNkMAIDCapOperation_Set*

Within the NkMAIDArray structure, the **ulType** member will be *kNkMAIDArrayType_Unsigned*, **wPhysicalBytes** will be four and **wLogicalBits** will be 32. The array will contain values from the eNkMAIDFilter enumeration.

9.26 kNkMAIDCapability_Prescan

The device will automatically set itself up for the original media.

Object types Image or thumbnail
ulType *kNkMAIDCapType_Process*
ulOperations *kNkMAIDCapOperation_Start*

9.27 kNkMAIDCapability_AutoFocus

The device will automatically set the focus of the device.

Object types Image or thumbnail
ulType *kNkMAIDCapType_Process*
ulOperations *kNkMAIDCapOperation_Start*

The module should update the value of the *kNkMAIDCapability_Focus* capability if it is able to.

9.28 kNkMAIDCapability_AutoFocusPt

This is the point that the module focuses upon.

Object types Image or thumbnail
ulType *kNkMAIDCapType_Point*
ulOperations *kNkMAIDCapOperation_Get, kNkMAIDCapOperation_Set*

The module is not required to provide this capability if it does not support focusing on a single point.

9.29 kNkMAIDCapability_Focus

This is the focus position of the device.

Object types Image or thumbnail
ulType *kNkMAIDCapType_Enum* or *kNkMAIDCapType_Range*
ulOperations *kNkMAIDCapOperation_Get, kNkMAIDCapOperation_GetArray, kNkMAIDCapOperation_Set*

Within the *NkMAIDEnum* structure, the **ulType** member will be *kNkMAIDArrayType_Float*.

9.30 kNkMAIDCapability_Coords

This is the target area to be acquired expressed as full resolution pixels.

Object types Image or thumbnail
ulType *kNkMAIDCapType_Rect*
ulOperations *kNkMAIDCapOperation_Get, kNkMAIDCapOperation_Set, kNkMAIDCapOperation_GetDefault*

The default value will be the largest area than can be acquired.

9.31 kNkMAIDCapability_Resolution

This is the acquire resolution in pixels/inch.

Object types Image or thumbnail
ulType *kNkMAIDCapType_Enum* or *kNkMAIDCapType_Range*
ulOperations *kNkMAIDCapOperation_Get, kNkMAIDCapOperation_GetArray, kNkMAIDCapOperation_Set*

Within the *NkMAIDEnum* structure, the **ulType** member will be *kNkMAIDArrayType_Float*.

9.32 kNkMAIDCapability_Preview

This will set a priority on speed or quality.

Object types Image or thumbnail
ulType *kNkMAIDCapType_Boolean*
ulOperations *kNkMAIDCapOperation_Get, kNkMAIDCapOperation_Set*

If the client sets this capability to TRUE, the module should try to acquire as quickly as possible. If the client sets it to FALSE, the module should try to produce the best quality possible.

9.33 kNkMAIDCapability_Negative

This will select the type of original media as either negative or positive.

Object types Image or thumbnail
ulType *kNkMAIDCapType_Boolean*
ulOperations *kNkMAIDCapOperation_Get; kNkMAIDCapOperation_Set*

If the module does not provide this capability, the client will make no assumptions about the original media.

9.34 kNkMAIDCapability_ColorSpace

This will select the color space of the data delivered to the client.

Object types Image or thumbnail
ulType *kNkMAIDCapType_Enum*
ulOperations *kNkMAIDCapOperation_Get; kNkMAIDCapOperation_GetArray; kNkMAIDCapOperation_Set*

Within the NkMAIDEnum structure, the **ulType** member will be *kNkMAIDArrayType_Unsigned* and **wPhysicalBytes** will be four. The enumeration will contain one or more values from the eNkMAIDColorSpace enumeration.

9.35 kNkMAIDCapability_Bits

This will select the number of bits to acquire per color.

Object types Image or thumbnail
ulType *kNkMAIDCapType_Enum*
ulOperations *kNkMAIDCapOperation_Get; kNkMAIDCapOperation_Set; kNkMAIDCapOperation_GetArray*

If the module does not provide this capability, the client will assume that eight bits per color will be acquired.

9.36 kNkMAIDCapability_Planar

This will select or merely report the transfer mode supported by the object.

Object types Image or thumbnail
ulType *kNkMAIDCapType_Boolean*
ulOperations *kNkMAIDCapOperation_Get; possibly kNkMAIDCapOperation_Set*

If the module only wishes to transfer the data as either planar or interleaved, it will not support the *kNkMAIDCommand_CapSet* command.

9.37 kNkMAIDCapability_Lut

This is a set of look up tables to be applied to the image data before it is transferred to the client.

Object types Image or thumbnail
ulType *kNkMAIDCapType_Array*
ulOperations *kNkMAIDCapOperation_Get; kNkMAIDCapOperation_Set; kNkMAIDCapOperation_GetArray*

Within the NkMAIDArray structure, the **ulType** member will be *kNkMAIDArrayType_Unsigned*. For color images, the array will be two or more look up tables, with the number of and order of the tables depending on the current color space. For RGB, there would be three tables, in the order of red, green, blue. For CMYK, there would be four tables, in the order of cyan, magenta, yellow, black. For monochrome images, there will be only one look up table.

9.38 kNkMAIDCapability_Transparency

This will select the type of original media as either transparent or reflective.

Object types Image or thumbnail
ulType *kNkMAIDCapType_Boolean*
ulOperations *kNkMAIDCapOperation_Get; kNkMAIDCapOperation_Set*

If the module does not provide this capability, the client will make no assumptions about the original media.

9.39 kNkMAIDCapability_Threshold

This is the threshold level for bilevel lineart images.

Object types Image or thumbnail
ulType *kNkMAIDCapType_Range*
ulOperations *kNkMAIDCapOperation_Get; kNkMAIDCapOperation_Set*

9.40 kNkMAIDCapability_Pixels

The number of pixels that will be acquired with consideration given to the current state of the data object.

Object types Image, thumbnail, or video
ulType *kNkMAIDCapType_Size*
ulOperations *kNkMAIDCapOperation_Get*

9.41 kNkMAIDCapability_ForceScan

This will determine whether unnecessary acquisitions (as determined by the module) will be performed by the device.

Object types Data object only
ulType *kNkMAIDCapType_Boolean*
ulOperations *kNkMAIDCapOperation_Get; kNkMAIDCapOperation_Set*

If this capability is set to TRUE, the device will always perform a physical scan when the *kNkMAIDCapability_Acquire* capability is started. If this capability is set to FALSE, the module may decide, based on its current state, whether such an acquisition is necessary. If a physical acquisition is not necessary, the module must go through the same steps that would normally be incurred during an acquisition (data delivery, I/O completion, etc.), except for the fact that the data would be supplied from an internal buffer as opposed to coming from a device. The default value for this capability is TRUE.

9.42 kNkMAIDCapability_ForcePrescan

This will determine whether unnecessary prescans (as determined by the module) will be performed by the device.

Object types Data object only
ulType *kNkMAIDCapType_Boolean*
ulOperations *kNkMAIDCapOperation_Get; kNkMAIDCapOperation_Set*

If this capability is set to TRUE, the device will always perform a physical prescan when the *kNkMAIDCapability_Prescan* capability is started. If this capability is set to FALSE, the module may decide, based on its current state, whether such a prescan is necessary. If a physical prescan is not necessary, the module must go through the same steps that would normally be incurred during a prescan (I/O completion, etc.). The default value for this capability is TRUE.

9.43 kNkMAIDCapability_ForceAutoFocus

This will determine whether unnecessary auto focus operations (as determined by the module) will be performed by the device.

Object types Data object only
ulType *kNkMAIDCapType_Boolean*
ulOperations *kNkMAIDCapOperation_Get; kNkMAIDCapOperation_Set*

If this capability is set to TRUE, the device will always perform a physical auto focus when the *kNkMAIDCapability_AutoFocus* capability is started. If this capability is set to FALSE, the module may decide, based on its current state, whether such an auto focus operation is necessary. If a physical auto focus is not necessary, the module must go through the same steps that would normally be incurred during an auto focus operation (I/O completion, etc.). The default value for this capability is TRUE.

9.44 kNkMAIDCapability_NegativeDefault

This is a source capability which will allow the default value of *kNkMAIDCapability_Negative* to be set.

Object types Source object only
ulType *kNkMAIDCapType_Unsigned*
ulOperations *kNkMAIDCapOperation_Get; kNkMAIDCapOperation_Set*

Once this capability is set, the source will use that value as the default value for *kNkMAIDCapability_Negative* for all image objects opened under this source thereafter. If this capability is not supported by the source, the module will use a reasonable default value for the items.

The module may change this capability in response to a change in the hardware. If the module initiates such a change, that change must be accompanied by a *kNkMAIDEvent_CapChange* event sent to the source.

9.45 kNkMAIDCapability_Firmware

This is a source capability which reports the firmware version of a device.

Object types Source object only
ulType *kNkMAIDCapType_String*
ulOperations *kNkMAIDCapOperation_Get*

This capability allows the client to read the firmware version of a device.

9.46 kNkMAIDCapability_CommunicationLevel1

This is a source capability which will allow the client to specify the method of communication to be used with the device.

Object types Source object only
ulType *kNkMAIDCapType_Enum*
ulOperations *kNkMAIDCapOperation_Get; kNkMAIDCapOperation_GetArray; kNkMAIDCapOperation_GetDefault; kNkMAIDCapOperation_Set*

The module will determine what methods of communications it can support. The user will make the choice from the string array. The **ulType** member of the *NkMAIDEnum* structure will be *kNkMAIDArrayType_String* or *kNkMAIDArrayType_PackedString*. One such list, for example, may include the following strings: "COM1", "COM2", "COM3", "COM4", and "SCSI". Optionally, the module may also analyze the system and eliminate any methods of communication that are not supported on that system. Using the example above, "COM3" and "COM4" may be removed if the system does not have those comm ports available.

9.47 kNkMAIDCapability_CommunicationLevel2

This is a source capability which will allow the client to specify more detail about the method of communication to be used with the device.

Object types Source object only
ulType *kNkMAIDCapType_Enum*
ulOperations *kNkMAIDCapOperation_Get; kNkMAIDCapOperation_GetArray; kNkMAIDCapOperation_GetDefault; kNkMAIDCapOperation_Set*

This capability will consist of a list of strings which will further specify the method of communication to be used. The user will make the choice from the string array. The **ulType** member of the NkMAIDEnum structure will be *kNkMAIDArrayType_String* or *kNkMAIDArrayType_PackedString*. If the communication method selected in *kNkMAIDCapability_CommunicationLevel1* is "COM1", then a typical list for this capability might include the following strings: "Comm Speed 19,200", "Comm Speed 38,400", "Comm Speed 57,600", and "Comm Speed 115,200". Optionally, the module may also analyze the system and eliminate any methods of communication that are not supported on that system. If the method of communication is fully described in *kNkMAIDCapability_CommunicationLevel1* and no further information is needed, then the **ulElements** member of the NkMAIDEnum structure should be set to zero.

9.48 kNkMAIDCapability_BatteryLevel

This is a source capability which will report the level of the battery.

Object types Source object only
ulType *kNkMAIDCapType_Integer*
ulOperations *kNkMAIDCapOperation_Get*

If the device can use a battery, this capability should be supported. If the battery is in use at the time this capability is queried, the module should return an integer between 0 and 100, inclusive, which indicates the percentage of battery life remaining. If the battery is not in use at the time this capability is queried (e.g. an external power supply is attached), the module should return a value of -1.

9.49 kNkMAIDCapability_FreeBytes

This is a source capability which will report the number of bytes available in the internal memory of the device.

Object types Source object only
ulType *kNkMAIDCapType_Float*
ulOperations *kNkMAIDCapOperation_Get*

If the device can use some sort of internal storage (e.g. Compact Flash), this capability should be supported. This capability should report the number of available bytes as a positive integer value. A floating point value is used to allow for a higher upper limit.

9.50 kNkMAIDCapability_Freeltems

This is a source capability which will report the number of items that can be added to the device using the available internal memory and the current device settings.

Object types Source object only
ulType *kNkMAIDCapType_Unsigned*
ulOperations *kNkMAIDCapOperation_Get*

If the device can use some sort of internal storage (e.g. Compact Flash), this capability should be supported.

9.51 kNkMAIDCapability_Remove

This capability instructs the device to remove an object from its internal memory.

Object types Source, item, data object
ulType *kNkMAIDCapType_Process*
ulOperations *kNkMAIDCapOperation_Start*

If this capability is started for an item or data object, that object should be removed from the device. If this capability is started for a source object, all item objects and their corresponding data objects should be removed from the device.

9.52 kNkMAIDCapability_FlashMode

This is the current flash mode.

Object types Source, item, data object
ulType *kNkMAIDCapType_Enum*
ulOperations *kNkMAIDCapOperation_Get; kNkMAIDCapOperation_GetArray; kNkMAIDCapOperation_GetDefault; kNkMAIDCapOperation_Set*

Within the NkMAIDEnum structure, the **ulType** member will be *kNkMAIDArrayType_Unsigned* and **wPhysicalBytes** will be four. The enumeration will contain one or more values from the eNkMAIDFlashMode enumeration.

9.53 kNkMAIDCapability_ModuleType

This is the type of device for which this module is intended.

Object types Module object only
ulType *kNkMAIDCapType_Unsigned*
ulOperations *kNkMAIDCapOperation_Get*

This capability will return one or more bit values from the eNkMAIDModuleType enumeration. This will help the client determine if this module should be used, or perhaps which user interface to display.

9.54 kNkMAIDCapability_AcquireStreamStart

This will start a stream acquire.

Object types Data object only
ulType *kNkMAIDCapType_Process*
ulOperations *kNkMAIDCapOperation_Start*

The module will begin calling the data object's data delivery callback with data. This process is designed to continue until the client stops it, so the progress callback should not be called.

9.55 kNkMAIDCapability_AcquireStreamStop

This will stop a stream acquire.

Object types Data object only
ulType *kNkMAIDCapType_Process*
ulOperations *kNkMAIDCapOperation_Start*

If a stream acquire is not in progress, the module will return *kNkMAIDResult_UnexpectedError*.

9.56 kNkMAIDCapability_AcceptDiskAcquisition

This capability is used to inform the module that it may deliver files on disk in response to *kNkMAIDCapability_Acquire*.

Object types Source object only
ulType *kNkMAIDCapType_Generic*
ulOperations *kNkMAIDCapOperation_Get, kNkMAIDCapOperation_Set*

When the client requests image data (with *kNkMAIDCapability_Acquire*), the module should not deliver disk files unless this capability has been called by the client with a non-null parameter. The client may use this capability to set a disk location. Once this is done, the module may deliver data by disk file, instead of by memory. After the disk file has been written and closed, the module should call the data object's data delivery callback. For Windows, the *pData* parameter should point to an *NkMAIDString* structure, which will contain the complete path (but not the name) for the new file. For Macintosh, the *pData* parameter should point to an *FSSpec* structure, which will indicate a folder for the new file. The module will choose a unique file name in the specified folder. Once the data delivery callback is called, the module should not access this file again for any reason. While preparing and writing the file to disk, the module may call the data object's progress callback if the acquisition will take a significant amount of time.

9.57 kNkMAIDCapability_Version

This is the version of the MAID specification which was followed in writing the current module.

Object types Module object only
ulType *kNkMAIDCapType_Unsigned*
ulOperations *kNkMAIDCapOperation_Get*

The client may determine the version of the MAID specification to which the module was written with this capability. This capability was introduced in MAID version 3.1. Therefore, if the module was written to a standard prior to 3.1, this capability will not be supported (will return *kNkMAIDResult_NotSupported*).

This capability will return a 4-byte unsigned value. The MAID version number will be broken into four parts, with the most significant part going into the most significant byte; the least significant part going into the least significant byte. For MAID version 3.1, for example, the most significant byte will contain 3, the next byte will contain 1, the next byte will contain 0, and the least significant byte will contain 0.

9.58 kNkMAIDCapability_FilmFormat

This will select the film format.

Object types Source object only
ulType *kNkMAIDCapType_Enum*
ulOperations *kNkMAIDCapOperation_Get, kNkMAIDCapOperation_GetArray, kNkMAIDCapOperation_Set*

The **ulType** member of the *NkMAIDEnum* structure will be *kNkMAIDArrayType_String* or *kNkMAIDArrayType_PackedString*. The enumeration will contain one or more values.

The client can select film format with this capability. For example, film format includes "35mm", "6*6" and "6*4.5". If the module supports only one format, this enumeration will contain only one value.

9.59 kNkMAIDCapability_TotalBytes

This is a source capability which will report the total number of bytes in the internal memory of the device.

Object types Source object only
ulType *kNkMAIDCapType_Float*
ulOperations *kNkMAIDCapOperation_Get*

If the device can use some sort of internal storage (e.g. Compact Flash), this capability should be supported. This capability should report the total number of bytes as a positive integer value. A floating point value is used to allow for a higher upper limit.

10 Function Definitions

10.1 MAID Entry Point Function

```
LONG MAIDEntryPoint(  
    LPNkMAIDObject    pObject,        // module, source, item or data object  
    ULONG              ulCommand,      // one of eNkMAIDCommand  
    ULONG              ulParam,        // parameter for the command  
    ULONG              ulDataType,     // one of eNkMAIDDataType  
    NKPARAM            data,           // pointer or long integer  
    LPNKFUNC           pfnComplete,    // function to call when complete, may be null  
    NKREF              refComplete    // passed to pfnComplete  
);
```

The return value will be one of eNkMAIDResult.

10.2 MAID Completion Function

```
void MAIDCompletion(  
    LPNkMAIDObject    pObject,        // module, source, item or data object  
    ULONG              ulCommand,      // one of eNkMAIDCommand  
    ULONG              ulParam,        // parameter for the command  
    ULONG              ulDataType,     // one of eNkMAIDDataType  
    NKPARAM            data,           // pointer or long integer  
    NKREF              refComplete,    // passed to MAIDEntryPoint  
    LONG              nResult          // one of eNkMAIDResult  
);
```

This is a placeholder for a callback function supplied by the client that will be called by the module after the command is complete. The parameters are the same parameters that were passed to the MAID entry point.

10.3 MAID Data Delivery Function

```
LONG MAIDData(  
    NKREF              refProc,        // reference set by client  
    LPNkMAIDDataInfo  pDataInfo,      // cast to LPNkMAIDImageInfo or LPNkMAIDSoundInfo  
    LPVOID             pData  
);
```

This is a placeholder for a callback function supplied by the client that will be called by the module to deliver data. The return value will be one of eNkMAIDResult.

10.4 MAID Event Notification Function

```
void MAIDEvent(  
    NKREF              refProc,        // reference set by client  
    ULONG              ulEvent,        // one of eNkMAIDEvent  
    NKPARAM            data            // pointer or long integer  
);
```

This is a placeholder for a callback function supplied by the client that will be called by the module to notify the client of events.

10.5 MAID Progress Notification Function

```
void MAIDProgress(
    ULONG          ulCommand,    // one of eNkMAIDCommand
    ULONG          ulParam,      // parameter for the command
    NKREF          refProc,      // reference set by client
    ULONG          ulDone,       // the numerator
    ULONG          ulTotal       // the denominator
);
```

This is a placeholder for a callback function supplied by the client that will be called by the module to notify the client of the progress of an asynchronous command.

For commands whose progress can be measured, the module will call this function at the start of the command with the **ulDone** parameter set to zero and the **ulTotal** parameter set to some positive value. When the command is complete, the module will call this function with the **ulDone** parameter equal to the **ulTotal** parameter. The module will call this function with the **ulDone** parameter set to zero and **ulTotal** exactly once each.

For commands whose progress cannot be measured, the module will call this function at the start of the command with the **ulDone** parameter set to one (1) and the **ulTotal** parameter set to zero. When the command is complete, the module will call this function with the **ulDone** parameter and the **ulTotal** parameter both equal to zero.

10.6 MAID User Interface Request Function

```
ULONG MAIDUIRequest(
    NKREF          refProc,      // reference set by client
    LPNkMAIDUIRequestInfo pUIRequest // information about the UI request
);
```

This is a placeholder for a callback function supplied by the client that will be called by the module to notify the user or ask the user a question. The **pObject** parameter is used if the UI Request contains capabilities to display to the user. If there are no capabilities to display, this parameter may be set to NULL. The **pUIRequest** parameter will point to an NkMAIDUIRequestInfo structure, which contains information about the message, the buttons, and optionally, which capabilities to display. The return value will be one of eNkMAIDUIRequestResult.

11 History

11.1 Changes Since v3.0 Revision 2

Added a Usage chapter.

Added this History chapter.

Wrote the Capabilities chapter.

Filled in the function descriptions in the Function Definitions chapter.

Added the `NkMAIDPoint` and `NkMAIDRect` structures.

Added the `eNkMAIDFilter` enumeration.

Changes to the `eNkMAIDResult` enumeration: added `kNkMAIDResult_Aborted`, `kNkMAIDResult_NoMedia`; removed `kNkMAIDResult_NotLocked`, `kNkMAIDResult_Locked`.

Changes to the `eNkMAIDCommand` enumeration: added `kNkMAIDCommand_Abort`; removed `kNkMAIDCommand_OpenModule`, `kNkMAIDCommand_GetChildCount`, `kNkMAIDCommand_GetChildIDs`; renamed `kNkMAIDCommand_OpenChild` to `kNkMAIDCommand_Open`, `kNkMAIDCommand_ClearToMark` to `kNkMAIDCommand_AbortToMark`.

Changes to the `eNkMAIDCapability` enumeration: added `kNkMAIDCapability_Children`, `kNkMAIDCapability_Start`, `kNkMAIDCapability_Prescan`, `kNkMAIDCapability_AutoFocus`, `kNkMAIDCapability_AutoFocusPt`, `kNkMAIDCapability_Preview`, `kNkMAIDCapability_Transparency`, `kNkMAIDCapability_Threshold`; removed `kNkMAIDCapability_Abort`, `kNkMAIDCapability_DataObj`; renamed `kNkMAIDCapability_DataAvailable` to `kNkMAIDCapability_DataTypes`; renamed `kNkMAIDCapability_Date` to `kNkMAIDCapability_DateTime`, `kNkMAIDCapability_AcquireMode` to `kNkMAIDCapability_Mode`, `kNkMAIDCapability_LightSource` to `kNkMAIDCapability_Filter`.

Changes to the `eNkMAIDDataType` enumeration: added `kNkMAIDDataType_PointPtr`, `kNkMAIDDataType_RectPtr`.

Changes to the `eNkMAIDArrayType` enumeration: added `kNkMAIDArrayType_Point`, `kNkMAIDArrayType_Rect`.

Changes to the `eNkMAIDCapType` enumeration: added `kNkMAIDCapType_Point`, `kNkMAIDCapType_Rect`.

Changed the comments of `kNkMAIDCommand_Mark` and `kNkMAIDCommand_AbortToMark`.

Changed the `ulObjectType` member of the `NkMAIDObject` structure to `ulType`.

11.2 Changes Since v3.0 Revision 3

Added a reference parameter to the data delivery, event notification and progress notification callback functions in the Function Definitions and Structures and Types chapters.

Wrote the Usage chapter.

Added definitions for `ULONG`, `NKPARAM`, `LPVOID`, `NKREF`, `LPMMAIDEntryPointProc`, `LPMMAIDCompletionProc`, `LPMMAIDDataProc`, `LPMMAIDEventProc`.

Changes to the `eNkMAIDDataType` enumeration: removed `kNkMAIDDataType_CharPtr`, `kNkMAIDDataType_ShortPtr`, `kNkMAIDDataType_BytePtr`, `kNkMAIDDataType_WordPtr`; renamed `kNkMAIDDataType_LongPtr` to `kNkMAIDDataType_IntegerPtr`, `kNkMAIDDataType_DwordPtr` to `kNkMAIDDataType_UnsignedPtr`.

Changes to the `eNkMAIDCapType` enumeration: removed `kNkMAIDCapType_Char`, `kNkMAIDCapType_Short`, `kNkMAIDCapType_Byte`, `kNkMAIDCapType_Word`; renamed `kNkMAIDCapType_Long` to `kNkMAIDCapType_Integer`, `kNkMAIDCapType_Dword` to `kNkMAIDCapType_Unsigned`.

There was no explanation of the `kNkMAIDCapability_DataTypes` capability in the Capabilities chapter.

11.3 Changes Since v3.0 Revision 4

Added chapter and section numbers.

Changes to the `eNkMAIDCapability` enumeration: added `kNkMAIDCapability_Pixels`, `kNkMAIDCapability_Stereo`, `kNkMAIDCapability_Samples`; renamed `kNkMAIDCapability_Size` to `kNkMAIDCapability_StoredBytes`.

Changes to the `eNkMAIDEvent` enumeration: renamed `kNkMAIDEvent_Add` to `kNkMAIDEvent_AddChild`, renamed `kNkMAIDEvent_Remove` to `kNkMAIDEvent_RemoveChild`.

Changes to the `eNkMAIDDataType` enumeration: added `kNkMAIDDataType_SizePtr`.

Changes to the `eNkMAIDArrayType` enumeration: added `kNkMAIDArrayType_Size`.

Changes to the `eNkMAIDCapType` enumeration: added `kNkMAIDCapType_Size`.

Added the `ulDimSize3` member to the `NkMAIDArray` structure and revised the structure description.

Added the `NkMAIDUIEventInfo`, `NkMAIDDataInfo`, `NkMAIDImageInfo` and `NkMAIDSoundInfo` structures.

11.4 Changes Since v3.0 Revision 5

Changes to Usage chapter: removed section “Child Addition and Removal”, added “Event Notification” and “User Interface Requests” sections.
 Changes to eNkMAIDEvent: added *kNkMAIDEvent_NewMedia*, *kNkMAIDEvent_MediaRemoved*; removed *kNkMAIDEvent_UserInterface*.
 Renamed the eNkMAIDUIEventType enumeration to eNkMAIDUIRequestType. Changed the description to fit the new usage.
 Renamed the eNkMAIDEventResult enumeration to eNkMAIDUIRequestResult. Changed the description to fit the new usage.
 Changes to eNkMAIDCommand: added *kNkMAIDCommand_EnumChildren*.
 Changes to eNkMAIDCapability: added *kNkMAIDCapability_UIRequestProc*, *kNkMAIDCapability_MediaPresent*.
 Changed the description of *kNkMAIDCommand_Open* in the Commands and Usage chapters.
 Added a User Interface Request function pointer type LPMAIDUIRequestProc.
 Changed the description and type name of the User Interface Request Structure in the Structures and Types chapter.
 Added a description for *kNkMAIDCommand_EnumChildren* to the Commands chapter.
 Changed the return value of the MAID Event Notification callback function from ULONG to void.

11.5 Changes Since v3.0 Revision 6

Changes to eNkMAIDResult: added *kNkMAIDResult_ZombieObject*, *kNkMAIDResult_OrphanedChildren*.
 Changes to eNkMAIDCommand: added *kNkMAIDCommand_GetParent*.
 Changes to eNkMAIDCapability: added *kNkMAIDCapability_Alive*, *kNkMAIDCapability_WarmedUp*.
 Changes to eNkMAIDEvent: added *kNkMAIDEvent_CapChange*, *kNkMAIDEvent_OrphanedChildren*, removed *kNkMAIDEvent_NewMedia*, *kNkMAIDEvent_MediaRemoved*.
 Changes to eNkMAIDUIRequestType: added *kNkMAIDUIRequestType_CustomOkCancel*.
 Explained more about references in Object Structure in the Structures and Types chapter.
 Explained more about prompt string in User Interface Request Structure in the Structures and Types chapter.
 Explained more about *kNkMAIDCommand_Close* in the Commands chapter.
 Explained more about *kNkMAIDCapability_Eject* in the Capabilities chapter.
 Added the Result Codes and Events chapters.
 Changed the NkMAIDRect structure from x1, y1, x2, y2 to x, y, w, h.
 Changed the **wBits** member of the NkMAIDImageInfo structure from a single value to an array of four values. Explained about its use in the Structures and Types chapter.

11.6 Changes Since v3.0 Revision 7

Changed parameter type in section 5.6 from LPNKFUNC to LPMAIDCompletionProc.
 Added the pObject parameter to the MAIDUIRequest function definition in section 10.6, and to the MAIDUIRequest typedef in section 5.11.
 Added the data parameter to the NkMAIDUIRequestInfo structure definition and a description of its use in section 5.22.
 Removed the *kNkMAIDUIRequestType_CustomOkCancel* value from the eNkMAIDUIRequestType enumeration in section 4.10.
 Added section 4.15, which describes the color space enumeration, eNkMAIDColorSpace.
 Changed *kNkMAIDCapability_Color* to *kNkMAIDCapability_ColorSpace* in the eNkMAIDCapability enumeration in section 4.14.
 Changed *kNkMAIDCapability_Color* to *kNkMAIDCapability_ColorSpace* in section 9.34.
 Changed the name of *kNkMAIDCapability_Alive* to *kNkMAIDCapability_IsAlive* in section 9.6.
 Changed the NkMAIDImageInfo structure in section 5.24 to accommodate various color spaces.

11.7 Changes Since v3.0 Revision 8

Reversed the History section so most recent history is at the end.
 Fixed several undefined references in document.
 Changed an enumeration reference from eNkMAIDDataType to eNkMAIDDataObjType in section 9.12.
 Added ulValueIndex and ulDefaultIndex members to the NkMAIDRange structure in section 5.19. Also change the name of nSteps to ulSteps.
 Added pObject member to the NkMAIDUIRequestInfo structure in section 5.22 and removed pObject from the MAIDUIRequest function definition in section 10.6.
 Removed the ulValue and ulDefault members from the NkMAIDArray structure in section 5.18. This structure is no longer used for enumerations.
 Added the NkMAIDEnum structure in section 5.26.
 Added *kNkMAIDDataType_EnumPtr* to the eNkMAIDDataType enumeration in section 4.3.
 Added *kNkMAIDCapType_Enum* to the eNkMAIDCapType enumeration in section 4.5.
 Added *kNkMAIDCapType_Enum* to the CapSet command in section 8.7.; to the CapGetDefault command in section 8.9; to the CapGetArray command in section 8.10.
 Changed the capability type of *kNkMAIDCapability_Children* to be *kNkMAIDCapType_Enum* in section 9.7.
 Changed the capability type of *kNkMAIDCapability_Mode* to be *kNkMAIDCapType_Enum* in section 9.18.
 Changed the capability type of *kNkMAIDCapability_SampleRate* to be *kNkMAIDCapType_Enum* in section 9.22.
 Changed the capability type of *kNkMAIDCapability_Filter* to be *kNkMAIDCapType_Enum* in section 9.25.
 Changed the capability type of *kNkMAIDCapability_Focus* to be *kNkMAIDCapType_Enum* in section 9.29.
 Changed the capability type of *kNkMAIDCapability_Resolution* to be *kNkMAIDCapType_Enum* in section 9.31.
 Changed the capability type of *kNkMAIDCapability_ColorSpace* to be *kNkMAIDCapType_Enum* in section 9.34.

11.8 Changes Since v3.0 Revision 9

Removed the LPNkMAIDObject parameter from the MAIDUIRequestProc function definition in section 5.11.
Added a description of capability groups to the introduction of section 9.
Modified kNkMAIDCapability_Lut to handle multiple color spaces in section 9.37.
Clarified the usage of the array member (data) in the User Interface Request Structure in section 5.22.
Corrected the example in section 3.4, which interchanged the commands kNkMAIDCommand_CapGet and kNkMAIDCommand_CapGetArray.
Specified that zero-based indexes are to be used in the NkMAIDRange structure (section 5.19) and in the NkMAIDEnum structure (section 5.26).
Specified that the eNkMAIDCapability enumeration should start with its first member at 1, instead of 0 in section 4.14.
Clarified the usage of the kNkMAIDEvent_CapChange event in section 7.5.
Added "data object" to the list of object types supporting kNkMAIDCapability_EventProc in section 9.3.
Removed kNkMAIDCapVisibility_Normal from the eNkMAIDCapVisibility enumeration in section 4.7.
Added the following values to the eNkMAIDResult enumeration in section 4.1: kNkMAIDResult_NoDataProc, kNkMAIDResult_OutOfMemory, kNkMAIDResult_UnexpectedError, and kNkMAIDResult_HardwareError.
Added descriptions for new result codes in sections 6.12, 6.13, 6.14, and 6.15.
Added "thumbnail" to the supported object types for the following capabilities in section 9: kNkMAIDCapability_Filter, kNkMAIDCapability_Prescan, kNkMAIDCapability_AutoFocus, kNkMAIDCapability_AutoFocusPt, kNkMAIDCapability_Focus, kNkMAIDCapability_Coords, kNkMAIDCapability_Resolution, kNkMAIDCapability_Preview, kNkMAIDCapability_Negative, kNkMAIDCapability_ColorSpace, kNkMAIDCapability_Bits, kNkMAIDCapability_Planar, kNkMAIDCapability_Lut, kNkMAIDCapability_Transparency, kNkMAIDCapability_Threshold, and kNkMAIDCapability_Pixels.

11.9 Changes Since v3.0 Revision 10

Added kNkMAIDResult_MissingComponent to the eNkMAIDResult enumeration in sections 4.1 and 6.16.
Added kNkMAIDCapability_ForceScan, kNkMAIDCapability_ForcePrescan, and kNkMAIDCapability_ForceAutoFocus to the eNkMAIDCapability enumeration in section 4.14.
Added descriptions for kNkMAIDCapability_ForceScan, kNkMAIDCapability_ForcePrescan, and kNkMAIDCapability_ForceAutoFocus in sections 9.41, 9.42, and 9.43, respectively.
Added an lpDetail member to the NkMAIDUIRequestInfo structure in section 5.22.
Added kNkMAIDDataType_GenericPtr to the eNkMAIDDataType enumeration in section 4.3.
Added kNkMAIDCapType_Generic to the eNkMAIDCapType enumeration in section 4.5.
Added kNkMAIDCapType_Generic to the CapGet, CapSet, and CapGetDefault commands in sections 8.7, 8.8, and 8.9, respectively.

11.10 Changes Since v3.0 Revision 11

Added a sentence to the introduction of section 8 describing the limitations on calling asynchronous commands from within a callback function.
Added a new capability, kNkMAIDCapability_NegativeDefault to the enumeration in section 4.14.
Added a new enumeration, eNkMAIDBooleanDefault, in section 4.16.
Added kNkMAIDDataType_BoolDefaultPtr to the list of data types in section 4.3.
Added kNkMAIDCapType_BoolDefault to the list of capability types in section 4.5.
Added the description of kNkMAIDCapability_NegativeDefault in section 9.44.

11.10 Changes Since v3.0 Revision 12

Removed kNkMAIDDataType_BoolDefaultPtr from the enumeration of data types.
Added a paragraph to section 10.5 to describe the "undefined" progress state.

11.11 Changes Since v3.0 Revision 13

Added enumeration eNkMAIDModuleTypes in section 4.17.
Added enumeration eNkMAIDFileDataTypes in section 4.18
Added fRemoveObject member to NkMAIDImageInfo and NkMAIDSoundInfo structures in sections 5.24 and 5.25.
Added structure NkMAIDFileInfo in section 5.27
Added command kNkMAIDCommand_ResetToDefault in section 8.16.
Added the following new capabilities: 9.45 kNkMAIDCapability_CommunicationLevel1; 9.46 kNkMAIDCapability_CommunicationLevel2; 9.47 kNkMAIDCapability_BatteryLevel; 9.48 kNkMAIDCapability_FreeBytes; 9.49 kNkMAIDCapability_FreeItems; 9.50 kNkMAIDCapability_Remove; 9.51 kNkMAIDCapability_FlashMode; 9.52 kNkMAIDCapability_ModuleType; 9.53 kNkMAIDCapability_AcquireStreamStart; 9.54 kNkMAIDCapability_AcquireStreamStop; 9.55 kNkMAIDCapability_AcceptDiskAcquisition; 9.56 kNkMAIDCapability_Version
Added the above capabilities to the eNkMAIDCapability enumeration in section 4.14.
Change location of MAID module files in section 3.1.

11.12 Changes Since v3.1 Revision 1

Changed type of `kNkMAIDCapability_NegativeDefault` from "BooleanDefault" to "Unsigned" in section 4.14.
Changed the definition of `kNkMAIDCapability_NegativeDefault` in section 9.44
Removed from `kNkMAIDBooleanDefault_None` from enumeration `eNkMAIDBooleanDefault`.

11.13 Changes Since v3.1 Revision 2

Added `kNkMAIDCapability_Firmware` to `eNkMAIDCapability` enum in section 4.14.
Renumbered sections 9.45 through 9.57 to 9.46 through 9.58.
Inserted `kNkMAIDCapability` description in section 9.45.

11.14 Changes Since v3.1 Revision 3

Added `kNkMAIDResult_VendorBase` to `eNkMAIDResult` enum in section 4.1.

11.15 Changes Since v3.1 Revision 4

Rearranged the values in `eNkMAIDFileDataTypes` in section 4.18 and added `kNkMAIDFileDataTypes_NIF`.
Added `kNkMAIDEvent_CapChangeValueOnly` to the `eNkMAIDEvent` enumeration in section 4.9.
Added a description of `kNkMAIDEvent_CapChangeValueOnly` in section 7.7.

11.16 Changes Since v3.1 Revision 5

Changed the description of `kNkMAIDCapability_AcceptDiskAcquisition` in section 9.56, so that the client specifies the destination folder but not the file name.
Added `kNkMAIDDataObjType_File` to the `eNkMAIDDataObjType` enumeration in section 4.2.
Restricted the `kNkMAIDDataObjType_File` value when using `kNkMAIDCapability_DataTypes` in section 9.12.
Added information about `kNkMAIDDataObjType_File` in section 5.23.

11.17 Changes Since v3.1 Revision 6

Added `kNkMAIDColorSpace_AppleRGB`, `kNkMAIDColorSpace_ColorMatchRGB`, `kNkMAIDColorSpace_NTSCRGB`, `kNkMAIDColorSpace_BruceRGB`, `kNkMAIDColorSpace_AdobeRGB`, `kNkMAIDColorSpace_CIERGB`, `kNkMAIDColorSpace_AdobeWideRGB`, `kNkMAIDColorSpace_NikonWideRGBg18` and `kNkMAIDColorSpace_NikonWideRGBg22` to the `eNkMAIDColorSpace` enumeration in section 4.15.

11.18 Changes Since v3.1 Revision 7

Changed the `AcceptDiskAcquisition` capability data type to "generic pointer" in section 9.56.
Added `kNkMAIDCapVisibility_Valid` to the `eNkMAIDCapVisibility` enumeration in section 4.7.

11.19 Changes Since v3.1 Revision 8

Changed the names of elements of `eNkMAIDColorSpace` in section 4.1.5. Changed `kNkMAIDColorSpace_NikonWideRGBg18` and `kNkMAIDColorSpace_NikonWideRGBg22` to `kNkMAIDColorSpace_AppleRGB_Compensated` and `kNkMAIDColorSpace_AdobeWideRGB_Compensated`, respectively.

11.20 Changes Since v3.1 Revision 9

Changed `kNkMAIDCapVisibility_Valid` to `kNkMAIDCapVisibility_Invalid` and changed the value to 0x0020.

11.21 Changes Since v3.1 Revision 10

Removed `kNkMAIDColorSpace_AdobeWideRGB_Compensated` from the `eNkMAIDColorSpace` enum.

11.22 Changes Since v3.1 Revision 11

Changed the `FlashMode` capability from a string-type enumeration to an integer enumeration in Section 9.52.
Added `eNkMAIDFlashMode` enumeration in section 4.19.

11.23 Changes Since v3.1 Revision 12

Added kNkMAIDFlashMode_SlowSyncRearCurtain in section 4.19.

11.24 Changes Since v3.1 Revision 14

Added kNkMAIDFileDataType_QuickTime in section 4.18.

11.25 Changes Since v3.1 Revision 15

Added kNkMAIDCapability_FilmFormat to eNkMAIDCapability enum in section 4.14.
Inserted kNkMAIDCapability description in section 9.58.

11.26 Changes Since v3.1 Revision 16

Added kNkMAIDCapability_TotalBytes to eNkMAIDCapability enum in section 4.14.
Inserted kNkMAIDCapability description in section 9.59.