Creative Labs, Inc.
1901 McCarthy Blvd.
Milpitas, CA 95035

Phone (408) 428 6600
Fax (408) 428 6611
www.creative.com

NOMAD Digital Audio Players

PC Software Development Kit

NOMAD Digital Audio Players

PC Software Development Kit

Revision History

3.5	26 Mar 2003	Added support for Jukebox 2/3/Zen family hierarchal data file handling
3.4	07 Aug 2002	Revised AddItem return value, ItemInfo attribute
3.3	26 Jul 2002	Revised ItemInfo structure, added new error codes
3.2	13 Nov 2001	Revised FormatInfo structure
3.1	01 Nov 2001	Revised Error Code Reference
3.0	02 Oct 2001	Unified SDK for NOMAD Jukebox and NOMAD II-series Players

While this is the first release of a unified SDK, it has been labeled as version 3.0 to distinguish it from prior releases of both NOMAD Jukebox SDK and NOMAD II SDK:

2.0	31 Aug 1999	*SDK for Creative NOMAD Digital Audio Player*
1.0		*SDK for Creative NOMAD Digital Audio Player*
1.0		*SDK for Creative NOMAD Jukebox Digital Audio Player*

Trademarks and Service marks

License Agreement, Limitation, and Disclaimer of Warranties

Restrictions on Use

Creative's BBS/FTP/website are the only online sites where users may download electronic files containing the manual and/or software,

Licensee shall use the manual and/or software only for the purpose of developing licensee applications compatible with Creative’s Nomad series of products, unless otherwise agreed to by further written agreement from Creative Technology Ltd.; and,

Licensee shall not distribute or copy the manual for any reason or by any means (including in electronic form) or distribute, copy, modify, adapt, reverse engineer, translate or prepare any derivative work based on the manual or software or any element thereof other than for the above said purpose, without the express written consent of Creative Technology Ltd. Creative Technology Ltd. reserves all rights not expressly granted to licensee in this license agreement.

Limitation of Liability

In no event will Creative Technology Ltd, or anyone else involved in the creation, production, and/or delivery of this software product be liable to licensee or any other person or entity for any direct or other damages, including, without limitation, any interruption of services, lost profits, lost savings, loss of data, or any other consequential, incidental, special, or punitive damages, arising out of the purchase, use, inability to use, or operation of the software, and/or licensee application, even if Creative Technology Ltd. or any authorized Creative Technology Ltd. dealer has been advised of the possibility of such damages. Licensee accepts said disclaimer as the basis upon which the software is offered at the current price and acknowledges that the price of the software would be higher in lieu of said disclaimer. Some states do not allow the limitation or exclusion of liability for incidental or consequential damages so the above limitations and exclusions may not apply to you.

Information in this document is subject to change without notice. Creative Technology Ltd. shall have no obligation to update or otherwise correct any errors in the manual and software even if Creative Technology Ltd. is aware of such errors and Creative Technology Ltd. shall be under no obligation to provide to Licensee any updates, corrections or bug-fixes which Creative Technology Ltd. may elect to prepare.

Creative Technology Ltd. does not warrant that the functions contained in the manual and software will be uninterrupted or error free and Licensee is encouraged to test the software for Licensee's intended use prior to placing any reliance thereon.

Table of Contents

NOMAD Digital Audio Players

PC Software Development Kit

This document describes the Creative NOMAD PC SDK, version 3.0. The document contains a description of the NOMAD digital audio player system, an overview of the SDK, and instructions for setting up the SDK. You will also find annotated sample code that shows how to use the SDK in your own code, and an explanation of function dependency.

NOTE: If you would like to support devices other than the NOMAD line of digital audio players, check http://developer.creative.com for the SDK that corresponds to the product you would like to support.

The heart of the document is a detailed description of the classes `ICTJukebox2` and `ICTNOMAD2` and their member functions. Following that is a member function synopsis for quick reference, and a list of error codes that may be returned by the functions.

This document is intended for application developers who want their applications to work with the NOMAD family of digital audio players through the NOMAD PC SDK. To use the SDK, you should know Microsoft Visual C++ programming in the Microsoft® COM environment. Visual C++ and COM programming are extensive topics that we cannot cover in this document.

NOTE: The structures used in this SDK are not compatible with Visual Basic.

An Overview of the NOMAD Digital Audio Player System

Creative's NOMAD system lets users create and manage digital audio files on their PC, then transfer those files to a NOMAD player for playback away from the PC. The system description that follows provides the details you will need to use the NOMAD SDK. For more details, you can visit the NOMAD web site at http://www.nomadworld.com.

Hardware

This SDK supports the NOMAD II line of portable players (II, IIc, and II MG) as well as the NOMAD Jukebox player. First-generation NOMAD players were produced in limited quantities, and are not supported by this SDK.

The NOMAD II Player

The NOMAD II and the NOMAD II MG players are pocket sized battery-powered units. They may include internal memory, and come with an open slot for an additional SmartMedia card. They both include a front LCD display, a headphone jack, docking contacts, a set of controls, and a USB port.

A user can work in three modes using the player: MP3 audio file playback, voice file recording and playback, and FM radio tuning. The controls on the player select the mode. They also select MP3 or voice files, start and stop playback, start and stop recording, set FM tuning, control volume, select playback equalization, and can delete any voice or MP3 file.

NOMAD II players have a docking bay for memory expansion and accept a single SmartMedia card that may currently contain anywhere from 32 MB to 128MB of flash RAM. When a NOMAD player has a SmartMedia card inserted, its memory is divided into two sections: internal RAM (built-in) and external RAM (the SmartMedia). Both of these sections can store MP3 files, voice files, or files of any other format—although the player itself will not be able to work with any files other than MP3, WMA, or voice files. Some NOMAD players have no internal RAM and depend completely on a SmartMedia card for their RAM memory.

SmartMedia is an industry standard for memory often used in digital cameras and other digital storage devices. You can use SmartMedia from other devices in the NOMAD player, but the card must be formatted using the SSFDC format before it will work. Formatting erases any existing files. Note that SSFDC-formatted SmartMedia may not work in other digital storage devices that may require their own special memory formatting.

SmartMedia cards are hot swappable in the NOMAD players. Because the cards retain files in memory even when removed from the player, you can use them to store MP3, WMA, and voice files outside the player, and then plug them in to work with the files. In that sense, they work as audio cassettes do for a cassette player.

The NOMAD Docking Station

Although both the NOMAD II and the NOMAD II MG players can download data directly from the PC through the USB cable, they can also download data from the PC through the docking station. The NOMAD docking station is powered by an AC adaptor, plugs into a

USB port, and has a docking bay for the NOMAD player. When the player is inserted into the docking station, the docking station recharges the player's batteries (you must use rechargeable batteries). Because of design differences between the NOMAD II and the NOMAD II MG, two docking stations are available.

The NOMAD Jukebox Player

The NOMAD Jukebox is sized like a portable CD player. It offers a substantially larger storage capacity with at least a 6 GB IDE hard disk drive. It can be powered by either batteries or a DC adaptor. It includes a front LCD display, a headphone jack, one or two line-out jacks, one line-in jack, and a set of controls. NOMAD Jukebox has a USB port for connecting to the computer.

The NOMAD Jukebox's internal software arranges and plays the music files (called tracks) into categories based on its ID3 tag attributes such as Album, Artist, Genre, etc. In addition, playlists can also be created to allow a user-defined list of tracks and play order.

The player currently plays WAV, MP3 and WMA files. As the internal software can be upgraded, new types of audio formats can be added through a firmware upgrade.

Software

The NOMAD system comes with two pieces of software: the player's internal software and an application that runs under Windows 98, Windows ME, or Windows 2000 on the PC.

Internal Player Software

The player's internal software responds to user input through player controls. Through multi-level menus, the user can navigate the music categories and select files to play. The NOMAD players can also record analog audio from their line input jacks into PCM-encoded WAV files. The NOMAD SDK takes care of all communication with the internal player software, so you do not need to know any of its specifics.

PlayCenter 2

PlayCenter 2 is a Windows application that the user must install on their PC before they can connect the NOMAD player and manage files there. PlayCenter 2 can create and play back digital audio directly on the PC. It can also download the files to the NOMAD player and upload the files back to the PC. Of course, this is subject to the copy protection restrictions imposed by the copyright owners.

Introducing the NOMAD PC SDK

The NOMAD PC SDK lets you add NOMAD player file-management control to your application. You can put most of PlayCenter 2's functions in your own application.

Visual C++ and the COM Environment

The NOMAD PC SDK was designed to work using Microsoft Visual C++ in the Microsoft COM environment. The SDK's API is implemented as a COM server class instantiated by your application. Each application on a PC using the NOMAD API has its own NOMAD COM object. The NOMAD COM object your application creates works with a single NOMAD COM server that serves all NOMAD clients. The server is implemented as a DLL.

When your application communicates with the NOMAD player, it does so through this chain:

- The application
- The NOMAD COM object
- The NOMAD DLL
- Appropriate drivers
- The USB port
- The NOMAD player

What the NOMAD SDK Can Do

The NOMAD API provides functions that perform many different tasks. They can:

- Detect connected NOMAD players and get their names.
- Get player's serial number and capabilities such as audio formats supported.
- Find total disk space and free disk space amounts.
- Download audio files to player and delete files there.
- Download non-audio files and delete them.
- Get audio and non-audio files out of the NOMAD player.
- Change the track's specific information such as artist, title, album, genre, etc.
- Create and remove playlists and edit and remove tracks in the playlist in NOMA Jukebox.
- Enumerate the list of categories in the NOMAD Jukebox player and list of tracks in each category.
- Get information about each track in the player.
- Manage the FM radio presets in NOMAD II players.

You will find much more about NOMAD SDK abilities in the sections that follow.

What's New in SDK 3.0

Version 3.0 of this SDK introduces a new interface that embraces all the features of the previous interface and adds additional features on top of them. Despite this, the number of functions in this new interface has not increased but instead reduced as a result of combining a number of functions into one.

The new features introduced by the new interface are:

- Get audio files out of the NOMAD to the PC.
- Transfer non-audio files in and out of NOMAD – enabling a secondary function of data storage in the NOMAD.
- Access to Jukebox properties like EAX and EQ settings.
- Remotely control playback of audio tracks in the Jukebox.
- Manage FM radio presets in NOMAD II player whenever the hardware supports.

Migrating from NOMAD SDK to Unified NOMAD SDK

With the introduction of the ICTNOMAD2 interface, several functions have been changed to accommodate cross-platform compatibility between NOMAD II/IIc/II MG and Jukebox players. The following is a table showing how to map the original NOMAD SDK functions to the new ICTNOMAD2 functions.

INomadPlayer, INomadPlayer2, INomadPlayer3 Functions	**ICTNOMAD2 Equivalent Functions**
GetNumberOfDevicesInstalled	GetDeviceCount
GetDeviceName	GetDeviceProperties(..,kDeviceNameString,..)
GetMemoryInfo	GetDeviceProperties(..,kStorageInfoStruct,..)
DownloadFile	AddItem
LaunchManager	No equivalent.

GetNumberOfFiles
GetNumberOfFilesEx1
	No equivalent but you can call a combination of FindFirstItem()/FindNextItem() to count the number of files. For example,
	`int nNumFiles=0;`
	`Hresult hr = FindFirstItem();`
	`while( hr == DAPSDK_SUCCESS )`
	`{`
	`nNumFiles++;`
	`hr = FindNextItem();`
	`}`

GetFileInfo GetFileInfoEx1	GetItemInfo
GetFormat	GetDeviceProperties(.., kAudioFormatInfoStruct, ..)
DeleteFile	RemoveItem
Format	SetDeviceProperties(.., kFormatStorage, ..)
SetParentHandle	SetCallbackWindow

Setting Up the NOMAD Jukebox SDK

To set up the NOMAD Jukebox SDK, you must first ensure that your PC meets the minimum hardware and software requirements. You can then install the NOMAD Jukebox SDK.

Minimum System Requirements

To use the NOMAD Jukebox SDK, you must have the following minimum hardware:

- PC with Intel Pentium 200 CPU or better
- 32 MB of RAM (64MB recommended)
- An available USB port
- A Creative NOMAD player

You will need the following minimum software:

- Microsoft® Windows® 98, Windows® Millennium Edition, or Windows® 2000
- Creative's NOMAD driver setup

Installing the NOMAD Jukebox SDK

To install the NOMAD SDK on your system, follow these steps:

1. Register the NOMAD COM to get it ready for development.

2. Unzip the sample source code into a project directory and create a subdirectory `\res` for its resource files. The sample code provides examples of how to use the NOMAD API.

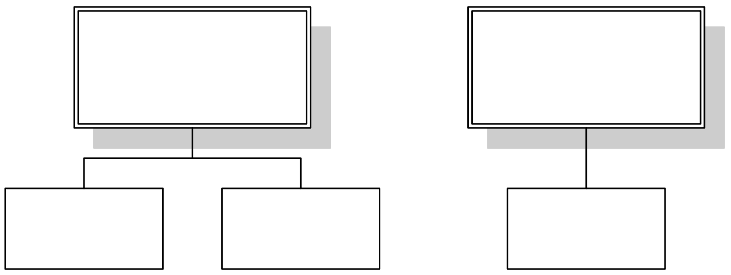
3. Use the files in the source directory for the definitions and dependency files required for developing an application using the NOMAD SDK.

Using the NOMAD SDK

To use the NOMAD SDK, your application must include the proper definition files and create a NOMAD COM object. It can then call the object's member functions to use the NOMAD API. Once the application is finished using the NOMAD API, it should release the NOMAD COM object. The following explanations and sample code show you how to do all this.

SDK 3.0 COM Object Description

This new SDK contains two new COM Class IDs (CLSIDs) CLSID_CTJukeBox2 and CLSID_CTNomad2. To maintain backward compatibility with NOMAD Jukebox SDK 1.0, two interface IDs (IID) are defined for the Jukebox.



IID_ICTJukebox

This interface is identical to that defined in NOMAD Jukebox SDK 1.0. The objective is to allow older applications to work immediately with this new SDK and to support future NOMAD Jukebox players by just re-compiling the application using the new header file from SDK 3.0. However, such applications cannot take advantage of the cool new features of this SDK, such as data file storage/retrieval and remote playback control. The developer is encouraged to use the second interface to access all these additional features.

Since the function and data definitions of this interface are identical to that in NOMAD Jukebox SDK 1.0, the use of these functions will not be described in this document. Developers who wish to use this interface should refer to the relevant document for descriptions of NOMAD Jukebox SDK 1.0.

IID_ICTJukebox2

This new interface encompasses all the features of interface one as well as the new features described earlier. Developers are encouraged to use the API of this interface.

IID_ICTNomad2

This new interface replaces and enhances INomadPlayer1, INomadPlayer2 and INomadPlayer3 interfaces in these ways:

- Non-blocking file transfer process – allowing applications to display their own progress user interface
- Allows files to be transferred from player to PC
- Exposes personalization settings like time and owner’s name
- Allows FM radio preset management
- Exposes firmware version query

Unified Command Structure

In an attempt to simplify the task of programming for devices in both the NOMAD II and Jukebox families, several commands now have a unified structure. The following chart details the functions that apply to each device family.

Player	Methods	Device Property	WM Messages

Jukebox Nomad II	FindFirstItem()
	FindNextItem()
	GetItemAttribute()
	AddItem()
	DeleteItem()
	GetItem()
	GetDeviceProperties()
	SetDeviceProperties()
		kDeviceSerialNumberValue
		kFirmwareVersion
		kDeviceNameString
		kPowerSourceValue
		kStorageInfoStruct
		kDateTimeStruct
		kOwnerNameString
		kAudioFormatCount
		kAudioFormatInfoStruct
		kLangEncodeSupport
			WM_DAPSDK_DOWNLOAD_PROGRESS
			WM_DAPSDK_DOWNLOAD_COMPLETE
			WM_DAPSDK_GETITEM_PROGRESS
			WM_DAPSDK_GETITEM_COMPLETE

Jukebox Only	Initialize2()
	ShutDown2()
	SetCallbackWindow2()
	GetDeviceCount2()
	CancelTransfer2()
	GetSDKVersion2()
	FindFirstRootItem()
	FindNextRootItem()
	FindFirstParentItem()
	FindNextParentItem()
	AddParentItem()
	DeleteParentItem()
	RenameParentItem()
	SetItemAttribute()
	AddItemsToParentItem()
	PlayControl()
	QueueControl()
		KhardwareVersion
		kAudioVolumeValue
		kAudioMuteValue
		kAudioEQPresetCount
		kAudioEQNameString
		kAudioEQAmountValue
		kAudioEQCurrentIndex
		kEAXCount
		kEAXNameString
		kEAXAmountValue
		kEAXCurrentIndex
			WM_DAPSDK_JUKEBOX_REMOVAL
			WM_DAPSDK_JUKEBOX_ARRIVAL
			WM_DAPSDK_PLAYLIST_COMPLETE
			WM_DAPSDK_PLAYBACK_COMPLETE
			WM_DAPSDK_PLAYBACK_ERROR
			WM_DAPSDK_PLAYBACK_PROGRESS

Nomad II Only	Initialize()
	ShutDown()
	SetCallbackWindow()
	GetDeviceCount()
	CancelTransfer()
	GetSDKVersion()
		kFMRadioPresetCount
		kFMRadioPresetValue
		kFormatStorage	

The enumerator of lType in _DAPSDK_ID structure is different for Jukebox SDK and Nomad II SDK.

```
typedef struct _DAPSDK_ID

{

    long    lID;           // stores the unique ID

    long    lType;         // stores the type (see enum above)

    BSTR    bstrName;      // stores the name

} DAPSDK_ID, *PDAPSDK_ID;
```

For Jukebox SDK:

```
enum

{

    kAudioTrackType = 1,  // this is a audio track

    kPlaylistType,     // this is a playlist

    kDataFileType      // this is a data file

};
```

For Nomad II SDK:

```
enum

{

    kInternalMemType = 0,  // this item resides in internal memory

    kExternalMemType,  // this item resides in external (removeable)
    media

};
```

Including Definition Files

A single header file is included:

- `nmsdk.h`

It contains the interfaces to the NOMAD Jukebox COM API and the constant definitions that the NOMAD Jukebox API uses. It also includes return codes and other constants. In addition, two other header files provided in Microsoft Visual Studio must also be included before `nmsdk.h`. They are:

- `objbase.h`
- `initguid.h`

To include these files in your project, follow these steps:

1. Copy the files to your project directory.

2. Include the `nmsdk.h` files in your main project .h source code. The following code snippet is an example:

   ```
   #include <Objbase.h>

   #include <initguid.h>

   #include "nmsdk.h" // CTNomad COM API definitions
   ```

3. You must also insert the necessary interface pointers in the project .h header file:

   ```
   IUnknown       * m_pUnknown;

   ICTJukeBox2    * m_pCTJukeBox2;
   ```

Creating the NOMAD COM Object

Once you have included the necessary files and defined appropriate pointers, you can create the NOMAD COM object in your application. The following steps walk you through it:

1. First initialize the pointers:

   ```
   m_pUnknown     = NULL;

   m_pCTNomad2    = NULL;   // NOMAD II object pointer

   m_pCTJukeBox   = NULL;  // NOMAD Jukebox object pointer
   ```

2. Next initialize OLE and the COM library:

   ```
   OleInitialize(NULL);
   ```

3. Now get the COM interface `IDigitalAudio`. If you require only one interface, you should get the `ICTJukeBox2` or ICTNomad2 pointer immediately:

   ```
   hr = CoCreateInstance(CLSID_CTJukeBox2, NULL, CLSCTX_INPROC_SERVER,

                         IID_ICTJukeBox2, (void**) &m_pCTJukeBox);
   ```

   or for NOMAD II devices,

   ```
   hr = CoCreateInstance(CLSID_CTNOMAD2, NULL, CLSCTX_INPROC_SERVER,

                         IID_ICTNomad2, (void**) &m_pCTNomad2);
   ```

   If you require more than one interface, you create an object of the inproc server and hold an unknown pointer of the object in `m_pUnknown`. A successful co-creation will automatically increase the reference count:

```
HRESULT hr = CoCreateInstance(CLSID_CTJukeBox2 /* CLSID_CTNOMAD2 */,

                       NULL, CLSCTX_INPROC_SERVER,

                       IID_IUnknown, (void**)&m_pUnknown);


    if(FAILED(hr) || m_pUnknown == NULL)

    {

        AfxMessageBox("Co Create Error");

        CString szTest;

        szTest.Format("%x",hr);

        AfxMessageBox(szTest);

        return FALSE;

    }

//successful QueryInterface auto increases ref count

hr = m_pUnknown->QueryInterface(IID_ICTJukeBox2,

                                 (void **)&m_pCTJukeBox);

/* for NOMAD II interface,

hr = m_pUnknown->QueryInterface(IID_ICTNomad2,

                                 (void **)&m_pCTNomad2); */

if(FAILED(hr))

{

    AfxMessageBox("Query interface error");

    return FALSE;

}
```

Calling the Member Functions

Once you have created the NOMAD COM object, you can use it by calling its member functions as shown in the following sample code. First, it checks to make sure that the NOMAD COM object `pICTJukeBox` or `pICTNomad2` is valid. Then, it initializes the NOMAD driver and queries to see how many different NOMAD devices are connected to the system.

```
BOOL CSampleDlg::COMInit()
{
    // get the number of device types
    long lCount = 0;
    if (m_pCTJukeBox == NULL)
        return FALSE;
    HRESULT hr = m_pCTJukeBox->Initialize2();    // interface 2 only
    /* for NOMAD II,
    HRESULT hr = m_pCTNomad2->Initialize();  */
    if(FAILED(hr))
    {
        AfxMessageBox("Initialize Failed");
        return FALSE;
    }
    HRESULT hr = m_pCTJukeBox->GetDeviceCount2(&lCount);
    /* for NOMAD II
    HRESULT hr = m_pCTNomad2->GetDeviceCount(&lCount);  */
    if(FAILED(hr))
    {
        AfxMessageBox("GetDeviceCount Failed");
        return FALSE;
    }
    m_nMaxNumOfDevice = lCount;
    if (m_nMaxNumOfDevice < 1)
        return FALSE;
    return TRUE;
}
```

You will find more information about the NOMAD COM object's member functions in a later section.

Releasing the COM Object

When your application is finished working with the NOMAD player and no longer needs the NOMAD COM object, the application should release it correctly to free its resources, and then release the OLE and COM libraries. The following sample code shows how it is done:

```
void CSampleDlg::COMRelease()

{

    if(m_pCTJukeBox)

        m_pCTJukeBox ->Release();  /* m_pCTNomad2->Release(); */


    if(m_pUnknown)

        m_pUnknown->Release();


    m_pUnknown      = NULL;

    m_pCTJukeBox    = NULL;


    //Release OLE + COM library

    OleUninitialize();

}
```

Using NOMAD API Functions

Now that you've seen the framework you use for creating a NOMAD COM object, calling its member functions, and then releasing the object when you're finished with it, let's take a closer look at how you use the member functions.

Programming for the NOMAD Jukebox

The NOMAD Jukebox is a portable music storage and playback device with a large-capacity hard disk built in. If the digitized audio is stored in a compressed music format (for example MP3), it can store over 100 hours of music at near-CD quality (128 Kbps of MP3, or equivalent). Assuming a song track averages about 6 minutes, which makes for a whopping 1,000 music tracks in a NOMAD Jukebox! As such, the music tracks have to be optimally organized so that the user can find the desired tracks easily.

The music tracks in NOMAD Jukebox are organized according to a few categories that are well known in the music world—Album titles, artist's names, and music genre or styles. These are generally known as ID3 information.

In addition to these categories, users may also collect a list of music tracks they want to order and save into a playlist. Playlists are containers of pointers to tracks in the NOMAD Jukebox and they can be seen on the device's LCD and selected for playback.

In SDK version 2.0, the storage types are no longer limited to music tracks. The user can copy any kind of data files into NOMAD Jukebox. This effectively makes the NOMAD Jukebox a handy temporary storage device. The NOMAD Jukebox SDK aims to expose these capabilities through a set of API described here.

Unlike SDK version 1.0, which deals with music tracks and playlists as separate entities, this version of the SDK will treat music tracks, playlists, and other non-music files as items. Items are the most basic entity.

The music library in the Jukebox is modeled in this SDK as a hierarchy of Root Items, Parent Items, and Items, as depicted here:

```
|- Root Item 1
|       |
|       |_______ Parent Item 1
|       |                   |_______ Item 11
|       |                   |_______ Item 12
|       |                   |_______ -
|       |                   |_______ -
|       |                   |_______ Item 1n
|       |
|       |_______ Parent Item 2
|       |                   |_______ Item 21
|       |               -
|       |               -
|       |               -
|       |               -
|       |
|       |_______ Parent Item m
|                           |_______ Item m1
|                           |_______ Item m2
|                           |_______ -
|                           |_______ Item mn
|
|- Root Item 2
|
|- Root Item 3
|
|       -
|       -
|       -
|- Root Item n
```

Root items are top-level category names such as Albums, Artists, Genre, Data, and Playlists. These correspond to the first level menu items on the Jukebox LCD. They can neither be removed nor renamed.

Under each root item are parent items that correspond to the album names, artist names, genre names or playlist names. All Parent Item types, except playlist, cannot be removed and renamed through the SDK API. Under each parent item are the items that belong to the respective parent items.

Items are the tracks or files corresponding to the collection such as album or artist, entries in a playlist or non-music files under the “Data” category. Non-music files, or data files, are somewhat different in that there are no real parent items. The only one called “Data 1” is really just a placeholder to complete the tree hierarchy.

Here is the tree layout of a typical NOMAD Jukebox library:

```
|- ALBUMS
|       |
|       |_______ Album 1
|       |               |_______ Track ab11
|       |               |_______ Track ab12
|       |               |_______ -
|       |               |_______ -
|       |               |_______ Track ab1n
|       |
|       |_______ Album 2
|       |               |_______ Track ab21
|       |       -
|       |       -
|       |       -
|       |       -
|       |
|       |_______ Album m
|                       |_______ Track abm1
|                       |_______ Track abm2
|                       |_______ -
|                       |_______ Track abmn
|
|- ARTISTS
|       |_______ Artist 1
|                       |_______ Track ar11
|                       |_______ Track ar12
|                       |_______ -
|                       |_______ Track ar1n
|
|- GENRES
|       |_______ Genre 1
|                       |_______ Track ge11
|                       |_______ Track ge12
|                       |_______ -
|                       |_______ Track ge1n
|
|- PLAYLISTS
|       |_______ Playlist 1
|                       |_______ Track pl11
|                       |_______ Track pl12
|                       |_______ -
|                       |_______ Track pl1n
|
|- DATA
        |_______ Data 1
                        |_______ Data file 1
                        |_______ Data file 2
                        |_______ -
                        |_______ Data file n
```

Programming for the NOMAD Jukebox 2/3/Zen

This family of hard disk based players has additional capabilities for the user to archive his non-audio files. This essentially allows the player's hard disk to be a multi-purpose storage device. These data files are stored differently from audio ones in two ways:

1. The files are not catalogued into album/artist/genre.

2. The files are not viewable from the player's LCD. The only way to access the files is via the PC.

Previous Jukebox models have the facility to store data files also, but were limited to storing them as a flat file list. The previous SDK's API therefore reflects this limitation. In the NOMAD Jukebox 2/3/Zen family, the device can organize the data in folders, just like a PC file system. Folders can be created, renamed, and deleted.

Programming for the NOMAD II Player

The NOMAD II player can either store files in its on-board memory (if any) or in the SmartMedia card.

Files in NOMAD II players, unlike those in NOMAD Jukebox are not categorized as different file types. The file system closely resembles that of a DOS file system. The properties of each file are limited to its file name, file size, creation date and time, and special attributes. The player recognizes audio files only through its file extension. For example, the player only plays *.MP3, *.WMA and *.NVF files.

Unlike NOMAD Jukebox, which has a music database with ID3 information built-in, the NOMAD II player reads the ID3 tags in the files to extract and display the song titles and artist names for display.

To maintain consistency with NOMAD Jukebox API, the same item ID structure is used to identify a file in NOMAD II.

Please note that both item type and item name are essential to uniquely identify a file.

Glossary of Terms Used in the API

Term	Definition
`Root Item`	The top-level categories of the NOMAD Jukebox library hierarchy, such as Albums, Artists, Genre, Playlists, and Data files. These items cannot be removed or renamed through the API.
`Parent Item`	The second (or middle) level category in the NOMAD Jukebox library. It is either an attribute of a music track in NOMAD Jukebox, such as the artist's name, album title, or the name of a playlist. These items can be removed or renamed. If a parent item is removed, all the (child) items under it are automatically removed. Right now, playlists are the only type of parent items that can be created through the API.
`Item`	The smallest entity in the library or storage. It can be a digitized music track or data file stored in NOMAD. Equivalent to a file in a storage device.
`Item ID`	A structure assigned to an item in NOMAD to uniquely identify it. This structure consists of an integer value ID, an item type constant and an item name string.
`Audio Track`	A file containing digitized audio such as WAV, MP3, or WMA. Also known as music tracks.
`Data File`	Any file that has non-audio content, i.e. files that cannot be played in the NOMAD Jukebox player, and will not appear in the player's LCD.
`Data Folder`	A Directory that contains non-audio files. This is for NOMAD Jukebox 2/3/Zen only.

Accomplishing NOMAD Tasks

Now that you have seen the API function dependencies, let us look at how to use the functions to accomplish some tasks with a NOMAD device. This brief description introduces you to the functions; for full details, read the section that follows, "Detailed API Description."

Setup

As you saw earlier, the first thing you need to do to work with the API is to create a NOMAD COM object for your application. For NOMAD Jukebox, you should then call these three functions immediately:

```
Initialize2()

GetDeviceCount2()

SetCallbackWindow2()
```

For NOMAD II, the corresponding functions are:

```
Initialize()

GetDeviceCount()

SetCallbackWindow()
```

They initialize the respective NOMAD libraries, check for installed NOMAD devices, and set the callback window handles in the respective NOMAD COM objects. Once set, you don't need to worry about using these three functions—unless you think a NOMAD device might be connected to or disconnected from the system while your application runs, in which case you should call `GetDeviceCount2()` or `GetDeviceCount()` whenever you think that might happen.

Getting Information about the NOMAD Player

In this new interface, all the information about the NOMAD player is available through a single function call:

```
GetDeviceProperties()
```

The `PropertyType` parameter should be set to the constant corresponding to the information you desire to query. At any time, you can query an attached NOMAD player to find out its device-specific information – like name, firmware and hardware version, and unique serial number.

For example, to ask for the player's firmware version:

```
GetDeviceProperties( .., kFirmwareVersion,..)
```

The complete list of property type constants can be found in the detailed description of this function under ***Detailed API Description*** section.

Querying NOMAD Audio Format Support

You can also query an attached NOMAD device to enumerate the formats supported and some details of each format, like the CODEC's descriptive name, maximum sampling rate, and number of channels supported. This will help you to decide which type of audio files to download to the NOMAD. To do so, you use these calls:

```
Long lFormatCount;

GetDeviceProperties(.., kAudioFormatCount, &lFormatCount);

If( lFormatCount )

{

    // allocate the format info structures

    FORMAT_INFO FormatInfo={0};

    for( int i=0; i<(int)lFormatCount; i++)

    {

GetDeviceProperties(.., kAudioFormatInfoStruct, &FormatInfo);

        // Do something…

    }

}
```

Querying NOMAD Storage Space

If you want information about the NOMAD Jukebox player's disk space, including how much is free to use, you can get it by calling:

```
GetDeviceProperties( .., kStorageInfoStruct, ..)
```

Due to the large size of the Jukebox's disk space (greater than 6 GB), the sizes are represented using two unsigned long integers, a high part and a low part.

This function will be applicable for NOMAD II players to get both the internal memory storage space as well as removable media (SmartMedia). You can call:

```
int MemType = kInternalMemType;   // change to kExternalMemType for
    SmartMedia memory

GetDeviceProperties( .., kStorageInfoStruct, &MemType ) // for
    internal memory
```

Setting Volume, Mute, and Equalizer (EQ)

This new SDK allows you to set the playback volume level and other audio controls such as EQ settings of NOMAD Jukebox. For example, to set the playback volume level, you can use:

```
long lVolInPercent = 50;  // set to 50%

SetDeviceProperties( .., kAudioVolumeLevel, &lVolInPercent);
```

You can also mute the audio output of the NOMAD Jukebox by:

```
long lMuteState = 1;  // TRUE = mute

SetDeviceProperties( .., kAudioMuteValue, &lMuteState);
```

Note that this feature is not applicable for NOMAD II players.

Querying and Setting EAX in NOMAD Jukebox

EAX is an audio enhancement technology exclusive to Creative and implemented in a wide range of audio products including NOMAD Jukebox. Due to the extensibility of the NOMAD Jukebox firmware, the number of effects cannot be hard-coded but instead has to be enumerated. Only one EAX preset is active at any one time.

To query the EAX settings available, you can do this:

```
long lNumEAX;

GetDeviceProperties( .., kEAXCount, &lNumEAX );

if( lNumEAX != 0 )

{

    TCHAR szEAXName[128];

    long lIndex;

    for( int n=0; n<lNumEAX, n++)

    {

        lIndex = n;

        GetDeviceProperties( .., kEAXNameString, (IUnknown *)&lIndex );

        GetStringFromBSTR((BSTR *)lIndex, szEAXName);

        // do something…

    }

    // get the current setting for EAX in player

    GetDeviceProperties( .., kEAXCurrentIndex, (Iunknown *)&lIndex);

    long lEAXPresetValue;

    GetDeviceProperties( .., kEAXAmountValue, &lEAXPresetValue );

    // do something..

}
```

To set the EAX to a certain preset, say index 1, and set it to 75%:

```
long lEAXPreset = 1;   // preset selected

SetDeviceProperties( .., kEAXCurrentIndex, (Iunknown *)&lEAXPreset );

lEAXPreset = 75;

SetDeviceProperties( .., kEAXAmountValue, (Iunknown *)&lEAXPreset );
```

Note that this feature is not applicable for NOMAD II players.

Item Identification for NOMAD Jukebox

Unlike SDK 1.0, the item ID in this SDK is no longer a simple long integer value. Instead, it is a structure defined in this form:

```
typedef struct {

    long lID;      // system-wide unique item ID number

    long lType;    // either audio track, playlist data file, or data
                   // folder

    BSTR bstrName; // item name string

} DAPSDK_ID;
```

This applies to root items and parent items as well. Note that these values are derived dynamically for each session and are not persistent. Therefore, the developer should not hard-code the item values.

Item Identification for NOMAD II Player

Unlike NOMAD Jukebox, the DAPSDK_ID structure members have slightly different meanings.

```
typedef struct {

    long lID;      // system-wide unique item ID number

    long lType;    // either internal or external(SmartMedia) memory

    BSTR bstrName; // file name string

} DAPSDK_ID;
```

In addition, there are no root items or parent items in NOMAD II and the functions that are related to these entities are not supported in ICTNomad2 interface.

Enumerating Root Items in NOMAD Jukebox

To get the list of root items in the NOMAD Jukebox, call these:

```
DAPSDK_ID ItemId;

// enumerate until no more

HRESULT hRes = FindFirstRootItem(0, (Iunknown *)&ItemId);

// store it or something…

While( hRes!= DAPSDK_E_ITEM_NOT_FOUND )

{

    hRes = FindNextRootItem(0, (Iunknown *)&ItemId);

    // store it or something..

        }
```

These calls will let you list out all major categories of the library in the player. The item type member in the ID structure indicates whether the root items are audio track categories, playlists, or data files. They will form the first level of a multi-level tree structure.

The ID structure returned from this call is required for subsequent calls to get the parent items.

Note that root items are not applicable to NOMAD II players.

Enumerating Parent Items in NOMAD Jukebox

To get the list of parent items of each root item in the NOMAD Jukebox, call these:

```
FindFirstParentItem(..,(Iunknown *)pRootItemId, (Iunknown *)&ItemId)

FindNextParentItem(..,(Iunknown *)pRootItemId, (Iunknown *)&ItemId)
```

One essential parameter required for these calls is root item ID obtained from earlier FindFirstRootItem() and FindNextRootItem() calls. These calls will let you list out all the values under each category of the library in the player. For example, if the root item is Album, the calls will yield a list of album titles. They will form the second level of a multi-level tree structure.

The ID structures returned from these calls are required for subsequent calls to get the item IDs.

Note that parent items are not applicable to NOMAD II players.

Enumerating Items in NOMAD Jukebox

To get the list of items in the NOMAD Jukebox, call these:

```
        FindFirstItem(..,(Iunknown *)pParentItemId, (Iunknown *)&ItemId)

        FindNextItem(..,(Iunknown *)pParentItemId, (Iunknown *)&ItemId)
```

These calls let you list out all the items under the specified parent (album title, artist name or genre category or playlists).

The calls return both the item name and the unique ID of each item. For a music track, it is a track title, and for a data file, it is the file name. This ID structure is required for subsequent calls to get or change information of the item.

Enumerating Data Items in NOMAD Jukebox 2/3/Zen

In NOMAD Jukebox 2/3/Zen players, folders can be created for non-audio files. The SDK user can create, delete, rename and set attributes to this type of items. The root item remains the same, Data, but no parent item exists. Instead, developers just begin enumerating items by calling `FindFirstItem/FindNextItem` using the root item as the parent item.

On returning from `FindFirstItem/FindNextItem` calls, the program is expected to check the lType field in the item ID structure to know the item is a data file or data folder. If the item is a data folder, the program can enumerate further by calling `FindFirstItem/FindNextItem` using the item ID as the parent item ID.

This method is very similar to Win32 API `FindFirstFile/FindNextFile` behavior.

Creating a Data Folder in NOMAD Jukebox 2/3/Zen

The program can create a data folder by calling `AddItem()` with data folder type, folder name, and specifying the parent folder ID in the item information structure.

For example, to create a folder called “New Folder”,

```
PBYTE pItemInfo = Build_Item_Info(

    “PARENT FOLDER”, kBINARY, lParentFolderId

                        // just the lID part of the DAPSDK_ID

    “FOLDER NAME”, Kascii, “New Folder”

);

AddItem( lDeviceID,

    KDataFolderType,    // item type == folder

    NULL,               // not required for folder creation

    LItemInfoSize,      // size of the item info buffer

    PItemInfo

    );
```

The new folder ID will be returned via `WM_DAPSDK_ADDITEM_COMPLETE` callback message. Note that when creating a folder, no progress callback messages will be received, unlike downloading a file.

Deleting a Data Folder in NOMAD Jukebox 2/3/Zen

To delete the data folder, call:

```
DeleteItem()
```

Note that the folder must be empty. The program must delete all sub-folders and files from this folder before it can be removed.

Changing Data File or Folder attributes in NOMAD Jukebox 2/3/Zen

The attributes of a data file or folder can be changed by calling:

```
SetItemAttribute()
```

This includes the folder or file name itself. To change the name of a file to “myname.xyz”,

```
Char szNewName[] = “MyName.xyz”;

SetItemAttribute(

    lDeviceID,

    pItemId,

    “FILE NAME”,       // attribute name

    kASCII,            // attribute type

    strlen(szNewName), // attribute data size in bytes

    (Iunknown *)szNewName

);
```

Note that the SDK will return an error if the file name already exists. To change the folder attribute to hidden:

```
DwCurrentAttr |= FILE_ATTRIBUTE_HIDDEN;

                       // follow Windows file attributes definition

SetItemAttribute(

    lDeviceID,

    pItemId,           // folder item ID

    “FILE ATTRIB”,     // attribute name

    kBINARY,           // attribute type

    sizeof(DWORD),     // attribute data size in bytes

    (Iunknown *)&dwCurrentAttr

);
```

Moving Data Files or Folders in NOMAD Jukebox 2/3/Zen

To move a data file or folder is to change the parent folder of a file or folder, using the `SetItemAttribute()` call. For example, to move a file identified by `FileId` to a folder identified by `DestFolderId`:

```
SetItemAttribute(

    lDeviceID,

    &FileId,

    "PARENT FOLDER",   // attribute name

    kBINARY,           // attribute type

    sizeof(DWORD),     // attribute data size in bytes

    (Iunknown *)&DestFolderId.lID

);
```

Enumerating Items in NOMAD II

To get the list of items (files) in the NOMAD II, call these:

```
FindFirstItem(..,NULL, (Iunknown *)&ItemId)

FindNextItem(..,NULL, (Iunknown *)&ItemId)
```

Note that the parent item ID argument is not used and should be set to NULL. These calls let you list out all the files in the storage media. To list files in the internal memory, set ***ItemId.lType*** to ***kInternalMemType*** and for SmartMedia, to ***kExternalMemType***.

The calls return both the file name and the unique ID of each item.

Downloading a File to NOMAD Jukebox

To download an audio or data file from the computer system to the NOMAD Jukebox, you can use the function:

```
AddItem()
```

For this call, one essential parameter for this call is a pointer to an item information structure. This is a variable-length structure with no restriction on the number of fields. For audio track type, title, CODEC type, playback length and file size are mandatory fields. For data files, only the file name and file size fields are required.

This call copies the contents of the specified file into the NOMAD Jukebox's hard disk. You can download music files (i.e. MP3, WMA, or WAV), or generic data files, into NOMAD Jukebox. However, the ***lItemType*** parameter has to be stated correctly.

For example, to download an MP3 file “mysong.mp3”,

```
PBYTE pItemInfo = Build_Item_Info(

        “TITLE”, “my song title”, // title

        “ALBUM”, “my song album”, // album

        “ARTIST”, “my song artist”, // artist

        “GENRE”, “Jazz”, // genre

        “CODEC”, “MP3”, // CODEC

        “LENGTH”, 170, // length in seconds

        “FILE SIZE”, GetFileSize(“mysong.mp3”) // file size in bytes

    );

DAPSDK_ID NewItemID;   // receive newly created track’s ID

AddItem ( .., kAudioTrackType, “mysong.mp3”,

            lItemInfoSize, (IUnknown *)pItemInfo,

            (IUnknown *)&NewItemID );
```

To download a non-audio file “myfile.xyz”:

```
PBYTE pItemInfo = Build_Item_Info(

        “FILE NAME”, “myfile.xyz”,

        “FILE SIZE”, GetFileSize(“myfile.xyz”)

    );

DAPSDK_ID NewItemID;   // receive newly created track’s ID

AddItem ( .., kDataFileType, “myfile.xyz”,

            lItemInfoSize, (IUnknown *)pItemInfo,

            (IUnknown *)&NewItemID );
```

Please refer to the sample program that comes with this SDK for an example of how to use this function.

Downloading a File to NOMAD II

To download a file from the PC to NOMAD II, you can use the function:

```
AddItem()
```

However, unlike NOMAD Jukebox, the ***lItemType*** parameter is not a file type, but target storage type, either internal memory (***kInternalMemType***) or SmartMedia card (***kExternalMemType***). In addition, the item information structure does not have the same attributes, as items in NOMAD Jukebox are not used.

Unlike the previous INomadPlayerX API, the download process is not blocking. The call to AddItem will return immediately before the download process is completed. The application is expected to monitor the download progress and completion through the callback messages.

Please refer to the sample program that comes with this SDK for an example of how to use this function.

Getting and Setting FM Radio Presets in NOMAD II

Some models of NOMAD II have built-in FM radio. There are 32 presets and they can be set through software. To get and set an FM radio preset,

```
WORD nCount=0;

GetDeviceProperties( .., kFMRadioPresetCount, (Iunknown *)&nCount );

if( nCount )   // supported

{

    // get the preset value of preset 1

    DWORD dwPreset;

    GetDeviceProperties( .., kFMRadioPresetValue, (Iunknown *)
    &dwPreset

    // .. do something to it,

    // set it to 98.7 MHz == 98700 kHz

    DAPSDK_RADIOPRESET preset = {0, 98700);

    SetDeviceProperties(.., kFMRadioPresetValue, (IUnknown *)&preset );

);
```

Deleting a File from NOMAD

To delete a file from the NOMAD's storage, call:

```
DeleteItem()
```

Getting a File from NOMAD

To transfer a file from the player to the PC, call:

```
GetItem()
```

One essential parameter is the destination file name, which the SDK will use to save the contents of the uploaded file.

Enumerating Playlists in NOMAD Jukebox

In SDK version 2.0, a playlist is a parent item. To get the list of playlists in the NOMAD Jukebox, you will need to first enumerate the root items to find the root item ID corresponding to playlists and call these:

```
// get the root item ID for Playlists

FindFirstParentItem(.., pPlaylistRootItemID,..)

FindNextParentItem(.., pPlaylistRootItemID,..)
```

These calls will return a unique ID for each playlist. This ID can be used in adding tracks into or removing tracks from the playlist, and in renaming or removing playlists.

Creating a New Playlist in NOMAD Jukebox

To create a new playlist, call:

```
AddParentItem(..,pPlaylistRootItemID, pNewPlaylistItemID )
```

passing in the playlist name as the parent item ID. This call will return a unique ID in the same parameter. Use this ID to add tracks into the playlist by calling:

```
AddItemsToParentItem(.., pNewPlaylistItemID, pAudioTrackItemList )
```

where `pAudioTrackItemList` is a pointer to an array of track item IDs.

Managing Playlists in NOMAD Jukebox

In this new interface, there is no function to remove tracks from a playlist. This is due to an internal limitation that makes it highly inefficient to remove individual entries in a playlist. Therefore, you are advised to make changes to a local copy of the playlist, delete the entire playlist, and create a new one from the changes made.

To remove a playlist altogether, call:

```
DeleteParentItem(.., pPlaylistItemID )
```

To rename a playlist, call:

```
RenameParentItem(.., pPlaylistItemID, bstrNewPlaylistItem )
```

In each of these calls, the playlist is identified by its ID. Therefore, you are required to enumerate the list of playlists before these calls can be executed.

Controlling Playback of Files in NOMAD Jukebox

To remotely control the playback of specific files, like selecting a track to play, pausing, or stopping a playback, use the function:

```
PlayControl()
```

The ***lPlayOperationType*** parameter should be set to the values corresponding to the action desired. These values are described at the end of this document.

The SDK also allows a single track to be queued for playback. This track will be played immediately after the currently playing track has completed playing. To queue a track in NOMAD Jukebox, do this:

```
QueueControl( .., kQueueTrack, TrackItemID )
```

To remove the queued track, do this:

```
QueueControl( .., kClearQueue, TrackItemID )
```

Detailed API Description

The NOMAD 3.0 API consists of 3 interfaces, `ICTJukeBox`, `ICTJukeBox2`, and `ICTNomad2`. ICTJukebox interface is identical to the interface in NOMAD Jukebox SDK 1.0 and exists here solely for the purpose of backward compatibility with older programs. The description of the interface and member functions are found in the document *SDK for Creative NOMAD Jukebox Digital Audio Player v1.0*, and will not be covered here. This section describes the ICTJukebox2 and ICTNomad2 interfaces and their member functions in detail.

Arguments described here are preceded by a ⮝ if they are an output argument—that is, a return value of the function. Arguments are preceded by a ⮟ if they are an input argument—that is, an argument to which you must provide a value. If an argument is preceded by both symbols, it is an input/output argument: you supply a value that the function reads; the function then overwrites the value with a new value for you to read.

The ICTJukeBox2 and ICTNomad2 Classes

The `ICTJukeBox2` class defines the NOMAD Jukebox COM object that an application creates to communicate with the NOMAD Jukebox. The `ICTNomad2` class defines the NOMAD II COM object for communicating with the NOMAD II player. Each NOMAD capable application has its own NOMAD COM object that communicates with the NOMAD player through a chain of components: the NOMAD system DLL, the NOMAD driver, and the USB port.

The member functions of the `ICTJukeBox2` class make up the NOMAD Jukebox API. You use them to accomplish tasks such as detecting NOMAD Jukebox, getting their device information, checking on NOMAD Jukebox's disk status, managing tracks and playlists in the NOMAD Jukebox, and more.

The member functions of the `ICTNomad2` class make up the NOMAD II API. You use them to accomplish tasks such as detecting NOMAD II, getting their device information, checking on NOMAD II's memory status, managing files in the NOMAD II, and more.

Constructor & Destructor

An application constructs and destructs `ICTJukeBox2` or `ICTNomad2` using standard COM conventions. That is, it calls `CoCreateInstance()` to create an `ICTJukeBox2` or `ICTNomad2` object and then calls `release()` on the object to destroy the object—all shown in the example code in "Using the Nomad SDK" earlier in this document.

Public Member Functions

These member functions of classes `ICTJukeBox2` and ICTNomad2 are public.

Initialize() / Initialize2()

This is the first function an application should call when using the NOMAD API. It loads and initializes the NOMAD system DLL.

```
HRESULT Initialize()

HRESULT Initialize2()
```

This function has no parameters.

The function may return one of these values:

Return Constant	Description
`S_OK`	The COM function executed successfully.
`DAPSDK_E_PLAYER_NOT_CONNECTED`	The function failed because no NOMAD device was detected.
`DAPSDK_E_LOAD_COMPONENTS_FAILED`	The function failed because one or more required files failed to load.
`DAPSDK_E_OUT_OF_MEMORY`	The function failed because there is not enough free memory available in the NOMAD player to complete this operation.
`E_FAIL`	The function failed for unspecified reasons.

Any returned values other than these are returned by the COM system.

Shutdown() / Shutdown2()

This is the last function an application should call when using the NOMAD API. It cleans up the internal data structures and unloads the NOMAD system DLL.

```
        HRESULT Shutdown()

        HRESULT Shutdown2()
```

This function has no parameters.

The function may return one of these values:

Return Constant	Description
`S_OK`	The COM function executed successfully.
`E_FAIL`	The function failed for unspecified reasons.

Any returned values other than these are returned by the COM system.

GetDeviceCount() / GetDeviceCount2()

It reports the number of different NOMAD II or NOMAD Jukebox devices that are connected to the system, a number you will use with all the other function calls in the API. It also assigns a device ID number to each device type so you can specify that device in your later API calls.

Please note that this ID number is not a unique identifier for the device. It is merely a number based on the number of devices connected to the PC. If the device is disconnected during the lifetime of your application, you should call this function again to identify the device.

```
HRESULT GetDeviceCount(long* lpCount );

HRESULT GetDeviceCount2(long* lpCount );
```

The function accepts these arguments:

Argument	**Description**
`IpCount`	⮙ This is a pointer to a variable where the function writes the number of NOMAD devices that are connected to the system.

The function may return these values:

Return Constant	**Description**
`S_OK`	The COM function executed successfully.
`DAPSDK_E_LOAD_COMPONENTS_FAILED`	The function failed because one or more required files failed to load.
`E_INVALIDARG`	The function failed to execute properly. One or more arguments are invalid or a NULL pointer.
`E_FAIL`	The function failed for unspecified reasons.

Any returned values other than these are returned by the COM system.

GetSDKVersion() / GetSDKVersion2()

This function returns the version of the NOMAD Jukebox SDK installed. The current version is 0x0300.

Note that before you use this function, you must call `Initialize2()` to load and initialize the internal data structures of the NOMAD Jukebox library DLL.

```
HRESULT GetSDKVersion ( IUnknown* lpSDKVersion );

HRESULT GetSDKVersion2 ( IUnknown* lpSDKVersion );
```

The function accepts these arguments:

Argument	**Description**
`lpSDKVersion`	⮝ This is a pointer to a structure DAPSDK_VERSION that contains the version of SDK installed in your system.

The function may return these values:

Return Constant	**Description**
`S_OK`	The COM function executed successfully.
`E_INVALIDARG`	The function failed to execute properly. One or more arguments are invalid or a NULL pointer.
`E_FAIL`	The function failed for unspecified reasons.

Any returned values other than these are returned by the COM system.

GetDeviceProperties()

GetDeviceProperties allows the calling application to query the NOMAD player's device-specific information. The format of the value returned depends on the property type requested.

```
GetDeviceProperties( long lDeviceID, long lPropertyType, IUnknown *
    lpValue );
```

The function accepts these arguments:

Argument	Description
`ldeviceID`	↓ This long integer value specifies a connected NOMAD device. The value should be in the range of from 1 to the number of devices returned by `GetDeviceCount()or GetDeviceCount2().`
`lpropertyType`	↓ This is one of the constant values corresponding to the property required to get from the player.
`lpValue`	↑↓ Address to a buffer containing value returned based on lPropertyType.

These are the possible values of lPropertyType:

Argument	Description and Return Value
`KdeviceSerialNumberValue`	Gets the 128-bit unique serial number of the NOMAD player or SmartMedia card. This is an SDMI-compliant ID. For NOMAD II players, pass in memory type, either kInternalMemType or kExternalMemType into lpValue to indicate whether to get the serial number of the device or the SmartMedia card. ***This value is ignored for NOMAD Jukebox.*** Returns a pointer to a 16-byte buffer containing the serial number in lpValue.
`kFirmwareVersion`	Gets the firmware version of NOMAD player. Returns a pointer to a DAPSDK_VERSION structure containing the firmware version information.
`kHardwareVersion`	Gets the hardware version of NOMAD Jukebox. Returns a pointer to a DAPSDK_VERSION structure containing the hardware version information. ***Not applicable for NOMAD II players.***
`kDeviceNameString`	Gets the internal descriptive name of NOMAD player. Returns a pointer to a BSTR string in lpValue containing the device's name.
`kDevicePowerSourceValue`	Get the current battery level in NOMAD Jukebox. Returns a pointer to BYTE value in lpValue containing the battery level in percentage left. ***Not applicable for NOMAD II players.***

Argument	Description and Return Value
`kStorageInfoStruct`	Gets the disk space information in NOMAD Jukebox or memory information for NOMAD II. For NOMAD II players, pass in memory type, either kInternalMemType or kExternalMemType into lpValue to indicate whether to get information about internal or SmartMedia memory. ***This value is ignored for NOMAD Jukebox.*** Returns a pointer to a DAPSDK_STORAGE_INFO structure in lpValue containing the disk/memory space information.
`kDateTimeStruct`	Gets the current Date and Time setting in NOMAD player. Returns a pointer to a DAPSDK_DATE_TIME structure.
`kOwnerNameString`	Gets NOMAD player's owner name. Returns a pointer to a BSTR string in lpValue.
`kAudioFormatCount`	Gets the number of audio formats (like WAV, MP3, WMA) supported by NOMAD player. Returns a pointer to a long integer in lpValue.
`kAudioFormatInfoStruct`	Details of the format supported, e.g., CODEC name, max sample rate, etc. Pass in the index of the format in lpValue Returns a pointer to a FORMATINFO structure in lpValue.
`kAudioVolumeValue`	Gets the current volume level of Jukebox player in percentage. Returns a pointer to a long integer in lpValue. ***Not applicable for NOMAD II players.***

Argument	Description and Return Value
`kAudioMuteValue`	Gets the current setting for the mute status of the NOMAD Jukebox. Returns a pointer to an unsigned short value in lpValue containing the mute status: 1="on" and 0="off". ***Not applicable for NOMAD II players.***
`kAudioEQPresetCount`	Gets the number of equalizer presets in the NOMAD Jukebox. Returns a pointer to a long integer value in lpValue containing the number of presets. ***Not applicable for NOMAD II players.***
`kAudioEQNameString`	Gets the name of the particular equalizer preset index. Pass in preset index in lpValue. Returns a pointer to BSTR in lpValue containing the name of the equalizer preset. ***Not applicable for NOMAD II players.***
`kAudioEQAmountValue`	Gets the level of the currently active equalizer preset. Pass in preset index in lpValue. Returns a pointer to a long integer in lpValue containing the current value in percentage. ***Not applicable for NOMAD II players.***
`kAudioEQCurrentIndex`	Query the currently active equalizer preset in the NOMAD Jukebox. Returns a pointer to a long integer in lpValue containing the preset index. ***Not applicable for NOMAD II players.***
`kEAXCount`	Gets the number of EAX presets available in the NOMAD Jukebox. Returns a pointer to a long integer value in lpValue containing the number of presets. ***Not applicable for NOMAD II players.***

Argument	Description and Return Value
`kEAXNameString`	Gets the name of the particular EAX preset index. Pass in preset index in lpValue. Returns a pointer to BSTR in lpValue containing the name of the EAX preset. ***Not applicable for NOMAD II players.***
`kEAXAmountValue`	Gets the level of the currently active EAX preset. Returns a pointer to a long integer in lpValue containing the current value in percentage. ***Not applicable for NOMAD II players.***
`kEAXCurrentIndex`	Query what the current EAX preset is in the NOMAD Jukebox. Returns a pointer to a long integer in lpValue containing the preset index. ***Not applicable for NOMAD II players.***
`KlangEncodeSupport`	Gets the language code pages supported in NOMAD player. Returns a pointer to an unsigned long in lpValue which contains bit-ordered values of one or more of the following: 0x00000001 – Latin 1 (CP1252) 0x80000000 – UTF-16
`kFMRadioPresetCount`	Gets the number of FM radio presets available in the NOMAD II player. Returns a pointer to an unsigned short value in lpValue containing the number of presets. ***Not all NOMAD II player models have FM radio built-in. If the return value is zero (0), this implies that this feature is absent in the player.*** ***Not applicable for NOMAD Jukebox players.***

Argument	Description and Return Value
`kFMRadioPresetValue`	Gets the FM radio preset value of this index. Pass in preset index in lpValue. Returns a pointer to an unsigned long integer in lpValue containing the current value in kHz. ***Not applicable for NOMAD Jukebox players.***

The function may return these values:

Return Constant	Description
`S_OK`	The COM function executed successfully.
`DAPSDK_E_LOAD_COMPONENTS_FAILED`	The function failed because one or more required files failed to load.
`DAPSDK_E_UNSUPPORTED`	The function failed because the property type specified is not supported for this function
`E_INVALIDARG`	The function failed to execute properly. One or more arguments are invalid or a NULL pointer.

SetDeviceProperties()

SetDeviceProperties allows the calling application to set some of the properties of the NOMAD players. Examples of such properties are the date and time, owner's name, volume, EQ and EAX presets, and FM radio presets.

```
SetDeviceProperties( long lDeviceID, long lPropertyType, IUnknown *
    lpValue );
```

The function accepts these arguments:

Argument	Description
`lDeviceID`	✓ This is a long integer value to specify a connected NOMAD device. The value should be in the range of 1 to the number of devices returned by `GetDeviceCount()` or `GetDeviceCount2()`.
`lPropertyType`	✓ This is one of the constant values corresponding to the property required to set in the player.
`lpValue`	✓ Address to a buffer containing the value to set.

These are the possible values of lPropertyType:

Argument	Description and Return Value
`kDateTimeStruct`	Sets the current Date and Time setting in NOMAD Jukebox. lpValue is a pointer to a NMJBSDK_DATE_TIME structure containing the new date and time.
`kOwnerNameString`	Sets NOMAD Jukebox owner's name. lpValue is a pointer to a BSTR string containing the new name.
`kAudioVolumeValue`	Sets the volume level in the NOMAD Jukebox player. lpValue is a pointer to a long integer containing the volume in percentage. ***Not applicable to NOMAD II players.***
`kAudioMuteValue`	Mutes/un-mutes NOMAD Jukebox audio output. lpValue is a pointer to a long integer value containing the mute status: 1="on" and 0="off". ***Not applicable to NOMAD II players.***
`kAudioEQCurrentIndex`	Sets the currently active equalizer preset in the NOMAD Jukebox. lpValue is a pointer to a long integer containing the preset index. ***Not applicable for NOMAD II players.***
`kAudioEQAmountValue`	Sets the level of the currently active equalizer preset. lpValue is a pointer to a long integer containing the value in percentage. ***Not applicable for NOMAD II players.***
`kEAXAmountValue`	Sets level of current EAX preset in NOMAD Jukebox. lpValue is a pointer to a long integer value containing the level in percentage (0-100). ***Not applicable to NOMAD II players.***

Argument	Description and Return Value
`kEAXCurrentIndex`	Sets the active EAX preset is in the NOMAD Jukebox. lpValue is a pointer to a long integer containing the preset index. ***Not applicable to NOMAD II players.***
`kFMRadioPresetValue`	Sets the FM radio preset value of a preset index in NOMAD II players. lpValue is a pointer to a DAPSDK_RADIOPRESET structure containing the preset index and its desired value in kHz. ***Not applicable to NOMAD Jukebox.***
`kFormatStorage`	Re-formats the media in NOMAD II player, removing all files. lpValue is a pointer to a long integer containing the memory type, either kInternalMemType (internal media) or kExternalMemType (SmartMedia card). ***Not applicable to NOMAD Jukebox.***

The function may return these values:

Return Constant	Description
`S_OK`	The COM function executed successfully.
`DAPSDK_E_LOAD_COMPONENTS_FAILED`	The function failed because one or more required files failed to load.
`DAPSDK_E_UNSUPPORTED`	The function failed because the property type specified is not supported for this function
`E_INVALIDARG`	The function failed to execute properly. One or more arguments are invalid or a NULL pointer.

SetCallbackWindow()/SetCallbackWindow2()

This function specifies the window handle with which the NOMAD COM will send transfer progress and status messages. Once the application sets the callback handle, the `ICTNomad2/ICTJukeBox2` object stores it for reference. If the application fails to set the handle, no status messages will be sent to your application.

Note that before you use this function, you must call `Initialize2()/Initialize()` and `GetDeviceCount2()/GetDeviceCount()` to detect connected devices, report their number, and then assign device IDs to each.

```
HRESULT SetCallbackWindow( long lDeviceID, long hWnd );

HRESULT SetCallbackWindow2( long lDeviceID, long hWnd );
```

The function accepts these arguments:

Argument	Description
`lDeviceID`	✓ This is a long integer value to specify a connected NOMAD device. The value should be in the range of 1 to the number of devices returned by `GetDeviceCount()` or `GetDeviceCount2()`.
`hWnd`	✓ This integer value is the window handle of the application calling this function. The window handle will be used by the NOMAD COM as target for sending status messages such as file transfer progress.

The function may return this value:

Return Constant	Description
`S_OK`	The COM function executed successfully.

Any returned values other than this are returned by the COM system.

FindFirstRootItem()

This function starts the Root Item enumeration process by querying the first Root Item. Subsequently you can use `FindNextRootItem()` to get the next Root Item.

Root items are top-level categories of the library in the NOMAD Jukebox. These include ARTIST, ALBUM, GENRE, PLAYLIST, and DATA.

Note that before you call either of these functions, you must first have called `GetDeviceCount2()` to detect and assign IDs to installed devices.

```
HRESULT FindFirstRootItem( long lDeviceID, IUnknown * lpRootItemID);
```

The functions accept these arguments:

Argument	Description
`lDeviceID`	▼ This is a long integer value to specify a connected NOMAD Jukebox device. The value should be in the range of from 1 to the number of devices returned by `GetDeviceCount2()`.
`lpRootItemID`	▲ This is a pointer to a `DAPSDK_ID` structure where the function writes the identifier to this Root Item.

The functions may return these values:

Return Constant	Description
`S_OK`	The COM function executed successfully.
`E_INVALIDARG`	The function failed to execute properly. One or more arguments are invalid or a NULL pointer.
`DAPSDK_E_ITEM_NOT_FOUND`	The function failed because there are no more items to find. End of enumeration.
`DAPSDK_E_PLAYER_NOT_CONNECTED`	The function failed because no NOMAD Jukebox device was detected.
`DAPSDK_E_LOAD_COMPONENTS_FAILED`	The function failed because one or more required files failed to load.
`DAPSDK_E_OUT_OF_MEMORY`	The function failed because there is not enough free memory available in the NOMAD Jukebox to complete this operation.
`E_FAIL`	The function failed for unspecified reasons.

Any returned values other than these are returned by the COM system.

FindNextRootItem()

This function continues the Root Item enumeration process by querying the next Root Item. You should keep calling this function until the return code `DAPSDK_E_ITEM_NOT_FOUND` is received.

```
HRESULT FindNextRootItem( long lDeviceID, IUnknown * lpRootItemID);
```

The functions accept these arguments:

Argument	Description
`lDeviceID`	⮟ This is a long integer value to specify a connected NOMAD Jukebox device. The value should be in the range of from 1 to the number of devices returned by `GetDeviceCount2()`.
`lpRootItemID`	⮝ This is a pointer to a `DAPSDK_ID` structure where the function writes the identifier to this Root Item.

The functions may return these values:

Return Constant	Description
`S_OK`	The COM function executed successfully.
`E_INVALIDARG`	The function failed to execute properly. One or more arguments are invalid or a NULL pointer.
`DAPSDK_E_ITEM_NOT_FOUND`	The function failed because there are no more items to find. End of enumeration.
`DAPSDK_E_PLAYER_NOT_CONNECTED`	The function failed because no NOMAD Jukebox device was detected.
`DAPSDK_E_LOAD_COMPONENTS_FAILED`	The function failed because one or more required files failed to load.
`DAPSDK_E_OUT_OF_MEMORY`	The function failed because there is not enough free memory available in the NOMAD Jukebox to complete this operation.
`E_FAIL`	The function failed for unspecified reasons.

Any returned values other than these are returned by the COM system.

FindFirstParentItem()

This function starts the Parent Item enumeration process by querying the first Parent Item for the Root Item. Subsequently you can use `FindNextParentItem()` to get the next Parent Item.

Parent items are second level categories in the library. For example, if the root item corresponds to ALBUM, the parent items are album names. If the root item is PLAYLIST, the parent items are playlist names. In the case of data files, there is only one parent item – “data 1”, which is just a placeholder.

Note that before you call this function, you must first have called `GetDeviceCount2()` to detect and assign IDs to installed devices. You must also call `FindFirstRootItem()/FindNextRootItem()` first to enumerate all the Root Items in order to obtain the appropriate Root Item ID for this call.

```
HRESULT FindFirstParentItem( long lDeviceID, long lRootItemUD, IUnknown
    * lpRootItemID, BSTR * lpRootItemName );
```

The functions accept these arguments:

Argument	**Description**
`lDeviceID`	⮟ This is a long integer value to specify a connected NOMAD Jukebox device. The value should be in the range of from 1 to the number of devices returned by `GetDeviceCount2()`.
`lRootItemID`	⮟ This is a pointer to a `DAPSDK_ID` structure containing a Root Item obtained from the previous Root Item enumeration process.
`lpParentItemID`	⮝ This is a pointer to a `DAPSDK_ID` structure where the function writes the identifier to this Parent Item.

The functions may return these values:

Return Constant	Description
`S_OK`	The COM function executed successfully.
`E_INVALIDARG`	The function failed to execute properly. One or more arguments are invalid or a NULL pointer.
`DAPSDK_E_ITEM_NOT_FOUND`	The function failed because there are no more items to find. End of enumeration.
`DAPSDK_E_PLAYER_NOT_CONNECTED`	The function failed because no NOMAD Jukebox device was detected.
`DAPSDK_E_LOAD_COMPONENTS_FAILED`	The function failed because one or more required files failed to load.
`DAPSDK_E_OUT_OF_MEMORY`	The function failed because there is not enough free memory available in the NOMAD Jukebox to complete this operation.
`E_FAIL`	The function failed for unspecified reasons.

Any returned values other than these are returned by the COM system.

FindNextParentItem()

This function continues the Parent Item enumeration process, beginning with a `FindFirstParentItem()` by querying the next Parent Item for a particular Root Item. You should keep calling this function until the return code `DAPSDK_E_KEY_NOTFOUND is received`.

```
HRESULT FindNextParentItem( long lDeviceID, long lRootItemUD,
    Iunknwon * lpParentItemID, BSTR * lpParentItemName );
```

The functions accept these arguments:

Argument	Description
`lDeviceID`	⋎ This is a long integer value to specify a connected NOMAD Jukebox device. The value should be in the range of from 1 to the number of devices returned by `GetDeviceCount2()`.
`lRootItemID`	⋎ This is a pointer to a `DAPSDK_ID` structure containing a Root Item obtained from the previous Root Item enumeration process.
`lpParentItemID`	⋏ This is a pointer to a `DAPSDK_ID` structure where the function writes the identifier to this Parent Item.

The functions may return these values:

Return Constant	Description
`S_OK`	The COM function executed successfully.
`E_INVALIDARG`	The function failed to execute properly. One or more arguments are invalid or a NULL pointer.
`DAPSDK_E_ITEM_NOT_FOUND`	The function failed because there are no more items to find. End of enumeration.
`DAPSDK_E_PLAYER_NOT_CONNECTED`	The function failed because no NOMAD Jukebox device was detected.
`DAPSDK_E_LOAD_COMPONENTS_FAILED`	The function failed because one or more required files failed to load.
`DAPSDK_E_OUT_OF_MEMORY`	The function failed because there is not enough free memory available in the NOMAD Jukebox to complete this operation.
`E_FAIL`	The function failed for unspecified reasons.

Any returned values other than these are returned by the COM system.

FindFirstItem()

This function starts the item enumeration process by querying the first Track for a Parent Item. Subsequently you can use `FindNextItem()` to get the next item.

For NOMAD Jukebox, each item corresponds to an audio track, a data file, or an entry in a playlist. You can use this function to enumerate the track list in a playlist. In this case, the Parent Item is a Playlist.

For NOMAD Jukebox 2/3/Zen, set the `lParentItemID` to either root item ID for Data to retrieve the first level folder or the appropriate item ID for the parent folder. The data item returned is either a file or a folder, depending on the lType member of the `DAPSDK_ID` structure.

For NOMAD II players, each item is a file and the parent item value is ignored.

Note that before you call this function, you must first have called `GetDeviceCount()` or `GetDeviceCount2()` to detect and assign IDs to installed devices. For NOMAD Jukebox, you must also call `FindFirstParentItem()/FindNextParentItem()` to enumerate all the Parent Items in order to obtain the appropriate Parent Item ID for this call.

```
        HRESULT FindFirstItem( long lDeviceID, Iunknown * lParentItemUD,
            IUnknown * lpItemID);
```

The functions accept these arguments:

Argument	Description
`ldeviceID`	⋎ This is a long integer value to specify a connected NOMAD device. The value should be in the range of 1 to the number of devices returned by `GetDeviceCount()` or `GetDeviceCount2()`.
`lparentItemID`	⋎ This is a pointer to a DAPSDK_ID structure containing a Parent Item obtained from previous Parent Item enumeration process. Set this value to NULL for NOMAD II devices.
`lpItemID`	⋏ This is a pointer to a `DAPSDK_ID` structure where the function writes the identifier to this Item.

The functions may return these values:

Return Constant	Description
`S_OK`	The COM function executed successfully.
`E_INVALIDARG`	The function failed to execute properly. One or more arguments are invalid or a NULL pointer.
`DAPSDK_E_ITEM_NOT_FOUND`	The function failed because there are no more items to find. End of enumeration.
`DAPSDK_E_PLAYER_NOT_CONNECTED`	The function failed because no NOMAD device was detected.
`DAPSDK_E_LOAD_COMPONENTS_FAILED`	The function failed because one or more required files failed to load.
`DAPSDK_E_OUT_OF_MEMORY`	The function failed because there is not enough free memory available in the NOMAD device to complete this operation.
`DAPSDK_E_UNSUPPORTED`	The function failed because the item type specified is not supported for this function
`E_FAIL`	The function failed for unspecified reasons.

Any returned values other than these are returned by the COM system.

FindNextItem()

This function continues the item enumeration process by querying the next item. You should keep calling this function until the return code `DAPSDK_E_ITEM_NOTFOUND is received`.

For NOMAD Jukebox, you can use this function to enumerate the track list in a playlist. In this case, the Parent Item is a Playlist.

For NOMAD Jukebox 2/3/Zen, set the `lParentItemID` to either root item ID for Data to retrieve the first level folder or the appropriate item ID for the parent folder. The data item returned is either a file or a folder, depending on the lType member of the `DAPSDK_ID` structure.

Note that before you call `FindNextItem()`, you must first have called `GetDeviceCount()` or `GetDeviceCount2()` to detect and assign IDs to installed devices.

```
HRESULT FindNextItem( long lDeviceID, long lParentItemID, IUnknown *
    lpItemID );
```

The functions accept these arguments:

Argument	Description
`lDeviceID`	⮟ This is a long integer value to specify a connected NOMAD device. The value should be in the range of 1 to the number of devices returned by `GetDeviceCount()` or `GetDeviceCount2()`.
`lParentItemID`	⮟ This is a pointer to a DAPSDK_ID structure containing a Parent Item obtained from previous Parent Item enumeration process.
`lpItemID`	⮝ This is a pointer to a DAPSDK_ID structure where the function writes the identifier to this Item.

The functions may return these values:

Return Constant	Description
`S_OK`	The COM function executed successfully.
`E_INVALIDARG`	The function failed to execute properly. One or more arguments are invalid or a NULL pointer.
`DAPSDK_E_ITEM_NOT_FOUND`	The function failed because there are no more items to find. End of enumeration.
`DAPSDK_E_PLAYER_NOT_CONNECTED`	The function failed because no NOMAD device was detected.
`DAPSDK_E_LOAD_COMPONENTS_FAILED`	The function failed because one or more required files failed to load.
`DAPSDK_E_OUT_OF_MEMORY`	The function failed because there is not enough free memory available in the NOMAD to complete this operation.
`DAPSDK_E_UNSUPPORTED`	The function failed because the item type specified is not supported for this function
`E_FAIL`	The function failed for unspecified reasons.

Any returned values other than these are returned by the COM system.

GetItemAttribute()

This function queries the attributes of an item. Attributes of an item are mostly provided by the user when creating a new track or added into an existing track in the NOMAD player. For NOMAD Jukebox, typical audio track attributes are artist's name, genre description, track length, album title, etc. For NOMAD II players, the attributes are its file size, file time and file attributes.

```
HRESULT GetItemAttribute( long lDeviceID, Iunknown * lpItemID, long
    lInItemInfoSize, long * lpOutItemInfoSize, Iunknown * lpItemInfo );
```

Since the function returns a variable length structure, you should call this function twice for each track. Call this function the first time to get the track information length by passing a NULL pointer as argument for `lpItemInfo`, which will return the required buffer size. Call this function the second time passing a pointer to a correctly sized buffer to receive the track information.

Here is a code fragment to illustrate this:

```
// get the track information structrure size first

GetItemAttribute( lDeviceId, lItemId, lBufSize, pulRequiredSize, NULL
    );

if( *pulRequiredSize )

{

    BYTE pItemInfo = new BYTE[*pulRequiredSize];

    // check for alloc success

GetItemAttribute( lDeviceId, lItemId, lBufSize, pulRequiredSize,
    pItemInfo );

    // decode pItemInfo…

}
```

Note that before you call this function either way, you must first have done the following:

- Call `GetDeviceCount()` or `GetDeviceCount2()` to detect and assign IDs to installed devices
- Query and enumerate the Root Items, Parent Items and items to get a valid item ID to pass in as argument
- For NOMAD II devices, the lType and bstrName members of the DAPSDK_ID structure are also required to identify the item

The format of an attribute structure is described in a section near the end of this document, called Structure Summary.

The functions accept these arguments:

Argument	Description
`lDeviceID`	⮟ This is a long integer value to specify a connected NOMAD device. The value should be in the range of 1 to the number of devices returned by `GetDeviceCount()` or `GetDeviceCount2()`.
`lpItemID`	⮟ This is a pointer to a DAPSDK_ID structure that specifies the ID of the item in NOMAD, obtained from previous call to `FindFirstItem()` or `FindNextItem()`.
`lInItemInfoSize`	⮟ This long integer specifies the size of the buffer pointed to in `lpItemInfo`.
`lpOutItemInfoSize`	⮝ This is a pointer to a long integer to receive the exact length of the item information buffer.
`lpItemInfo`	⮝ This is a pointer to a buffer to receive the item information structure. If you set this to NULL, the track information will not be returned but the required length will be set in `lpOutItemInfoSize`.

The functions may return these values:

Return Constant	Description
`S_OK`	The COM function executed successfully.
`E_INVALIDARG`	The function failed to execute properly. One or more arguments are invalid or a NULL pointer.
`DAPSDK_E_ID_INVALID`	The function failed because the Track or Playlist ID does not exist in the device.
`DAPSDK_E_ITEM_NOT_FOUND`	The function failed because there are no more items to find. End of enumeration.
`DAPSDK_E_PLAYER_NOT_CONNECTED`	The function failed because no NOMAD device was detected.
`DAPSDK_E_LOAD_COMPONENTS_FAILED`	The function failed because one or more required files failed to load.
`DAPSDK_E_OUT_OF_MEMORY`	The function failed because there is not enough free memory available in the NOMAD to complete this operation.
`E_FAIL`	The function failed for unspecified reasons.

Any returned values other than these are returned by the COM system.

SetItemAttribute()

This function updates the properties of each attribute of an item in NOMAD Jukebox by calling this function for the specified item.

You can change an existing item attribute or add a new item attribute to the item properties. Examples of audio track attributes are the album title, artist name, genre, or adding new attributes such as composer. The format of an attribute is described in a section near the end of this document, called Structure Summary.

For NOMAD Jukebox 2/3/Zen, you can use this function to rename data files and folders as well as to change their attributes.

This function is not applicable for NOMAD II devices.

```
HRESULT SetItemAttribute( long lDeviceID, Iunknown * lpItemID, BSTR
    bstrAttributeName, long lAttributeType, long lAttributeDataSize,
    IUnknown *lpAttributeData );
```

The function accepts these arguments:

Argument	**Description**
`lDeviceID`	✓ This is a long integer value to specify a connected NOMAD Jukebox device. The value should be in the range of from 1 to the number of devices returned by `GetDeviceCount2()`.
`lpItemID`	✓ This is a pointer to a DAPSDK_ID structure that specifies the ID of the item in NOMAD Jukebox, obtained from previous call to `FindFirstItem()` or `FindNextItem()`.
`bstrAttributeName`	✓ This is a string value of type BSTR containing the name of the attribute to set.
`lAttributeType`	✓ This long integer value specifies the type of the attribute to set. It is either 0 for ASCII, 1 for binary, or 2 for UTF-16.
`lAttributeDataSize`	✓ This long integer value specifies the length of the `lpAttributeData` argument.
`lpAttributeData`	✓ This is a pointer to a buffer that contains the track attribute to update.

The functions may return these values:

Return Constant	**Description**
`S_OK`	The COM function executed successfully.
`E_INVALIDARG`	The function failed to execute properly. One or more arguments are invalid or a NULL pointer.
`DAPSDK_E_ITEM_NOT_FOUND`	The function failed because there are no more items to find. End of enumeration.
`DAPSDK_E_PLAYER_NOT_CONNECTED`	The function failed because no NOMAD Jukebox device was detected.
`DAPSDK_E_LOAD_COMPONENTS_FAILED`	The function failed because one or more required files failed to load.
`DAPSDK_E_OUT_OF_MEMORY`	The function failed because there is not enough free memory available in the NOMAD Jukebox to complete this operation.
`E_FAIL`	The function failed for unspecified reasons.

Any returned values other than these are returned by the COM system

AddItem()

This function copies a file from a source indicated by the application into the NOMAD player. The file can be audio track or a data file. For the NOMAD Jukebox, the type of item has to be stated clearly, as different file types are stored differently in the player. The required arguments include the item information structure and full path to the file in the PC.

For NOMAD Jukebox 2/3/Zen, you can use this function to create data folders in the player. To do that, you'll need to put the parent folder ID into the variable-length structure `lpItemInfo`. The attribute name is "PARENT FOLDER", type BINARY, and data the `lID` part of `DAPSDK_ID`. Note that since folder creation is a very quick operation, there will not be any progress callbacks. Instead, only a single complete message will be sent.

For NOMAD II players, the item type is either internal memory or external memory (SmartMedia). The item information fields are not required and will be ignored.

```
HRESULT AddItem( long lDeviceID, long lItemType, BSTR bstrSrcFileName,
    long lItemInfoSize, IUnknown * lpItemInfo);
```

For audio tracks, this function also requires that a minimum set of fields be specified in the item information structure. They are track title, CODEC, and file size.

Note that this is a non-blocking function, i.e., the function call will return control immediately, before the file contents are downloaded. Download progress status will be sent to the calling application periodically via Windows' messaging mechanism. The messages are defined in the section NOMAD SDK Windows Messages. The window handle to receive these messages is the one set by calling `SetCallbackWindow2/SetCallbackWindow()`.

Note that this function cannot be used to add tracks to a playlist. This capability is available only in `AddItemsToParentItem()`.

The function accepts these arguments:

Argument	**Description**
`lDeviceID`	✓ This is a long integer value to specify a connected NOMAD device. The value should be in the range of 1 to the number of devices returned by `GetDeviceCount()` or `GetDeviceCount2()`.
`lItemType`	✓ This is a long integer value to specify the type of file to be downloaded to the NOMAD player. For NOMAD Jukebox, they can be either `kAudioTrackType` or `kDataFileType`. For NOMAD II, they are `kInternalMemType` or `kExternalMemType`.
`bstrSrcFileName`	✓ This is a string of type BSTR containing the full path of the audio file to download to the NOMAD device. Note that the COM server will free the memory allocated for `bstrSrcFileName` at the end of this function.
`lItemInfoSize`	✓ This is a long integer value to specify the size of the structure indicated in `lpItemInfo` argument. Set to NULL for NOMAD II.
`lpItemInfo`	✓ This is a pointer to a variable length structure containing the attributes of the file to be downloaded to the NOMAD Jukebox. Set to NULL for NOMAD II.

These are the possible callback messages:

Message	Description
`WM_DAPSDK_ADDITEM_PROGRESS`	Notifies the caller of transfer progress. wParam contains percent-completed value.
`WM_DAPSDK_ADDITEM_COMPLETE`	Notifies the caller of transfer completion. wParam contains the error status of the transfer.

The functions may return these values:

Return Constant	Description
`S_OK`	The COM function executed successfully.
`E_INVALIDARG`	The function failed to execute properly. One or more arguments are invalid or a NULL pointer.
`DAPSDK_E_FILE_ALREADY_EXISTS`	The function failed because a track with the same Album, Artist and Title already exists in the NOMAD Jukebox or a file with the same name already exist in NOMAD II. For NOMAD Jukebox, the ID (unsigned long integer) of the track that already exists is returned in TRACK ID attribute of the lpItemInfo argument. This is to allow the caller to remove and replace this track if desired.
`DAPSDK_E_TRACK_READ_WRITE_ERROR`	The function failed because an error occurred during the file transfer process.
`DAPSDK_E_LIB_BUSY`	The function failed because the NOMAD library DLL is currently busy with previous operation.
`DAPSDK_E_PLAYER_NOT_CONNECTED`	The function failed because no NOMAD device was detected.
`DAPSDK_E_LOAD_COMPONENTS_FAILED`	The function failed because one or more required files failed to load.
`DAPSDK_E_OUT_OF_MEMORY`	The function failed because there is not enough free memory available in the NOMAD device to complete this operation.
`DAPSDK_E_UNSUPPORTED`	The function failed because the item type specified is not supported for this function.
`E_FAIL`	The function failed for unspecified reasons.

Any returned values other than these are returned by the COM system.

GetItem()

This function retrieves or uploads a file and its contents in the NOMAD player to the PC. The file can be audio track or a data file. The required arguments include the item ID of the target file and the full path to the destination file in the PC.

Note that for some audio tracks, uploading to the PC is restricted due to copy restrictions imposed by the copyright vendor. Attempting to upload these files will return an error code `DAPSDK_E_TRACK_UPLOAD_DENIED`. This includes files that are protected with Microsoft DRM (WMA files).

Note that this is a non-blocking function, i.e., the function call will return control immediately, before the file contents are transferred. Transfer progress status will be sent to the calling application periodically via Windows' messaging mechanism. The messages are defined in the section NOMAD SDK Windows Messages. The window handle to receive these messages is the one set by calling `SetCallbackWindow2()/SetCallbackWindow()`.

```
HRESULT GetItem( long lDeviceID, BSTR bstrDestinationFileName, IUnknown
    * lpItemID);
```

The function accepts these arguments:

Argument	Description
`lDeviceID`	✓ This is a long integer value to specify a connected NOMAD device. The value should be in the range of 1 to the number of devices returned by `GetDeviceCount()` or `GetDeviceCount2()`.
`bstrDestinationFileName`	This is a string of type BSTR containing the full path of the file to save the uploaded file. Note that the COM server will free the memory allocated for `bstrDestinationFileName` at the end of this function.
`lpItemID`	✓ This is a pointer to a `DAPSDK_ID` structure of the ID of the target file in NOMAD player.

These are the possible callback messages:

Message	Description
`WM_DAPSDK_GETITEM_PROGRESS`	Notifies the caller of transfer progress. wParam contains the percent-completed value.
`WM_DAPSDK_GETITEM_COMPLETE`	Notifies the caller that the transfer is completed. wParam contains the error status of the transfer.

The functions may return these values:

Return Constant	Description
`S_OK`	The COM function executed successfully.
`E_INVALIDARG`	The function failed to execute properly. One or more arguments are invalid or a NULL pointer.
`DAPSDK_E_TRACK_READ_WRITE_ERROR`	The function failed because an error occurred during the file transfer process.
`DAPSDK_E_LIB_BUSY`	The function failed because the NOMAD library DLL is currently busy with previous operation.
`DAPSDK_E_PLAYER_NOT_CONNECTED`	The function failed because no NOMAD device was detected.
`DAPSDK_E_LOAD_COMPONENTS_FAILED`	The function failed because one or more required files failed to load.
`DAPSDK_E_OUT_OF_MEMORY`	The function failed because there is not enough free memory available in the NOMAD to complete this operation.
`DAPSDK_E_UNSUPPORTED`	The function failed because the item type specified is not supported for this function
`DAPSDK_E_ITEM_UPLOAD_DENIED`	The function failed because the file specified is not allowed to be uploaded to the PC due to copy protection.
`E_FAIL`	The function failed for unspecified reasons.

Any returned values other than these are returned by the COM system.

DeleteItem

This function removes a file in the NOMAD player.

For NOMAD Jukebox 2/3/Zen, you can use this function to delete data folders in the player. However, the folder has to be empty before it can be deleted.

```
HRESULT DeleteItem( long lDeviceID, IUnknown * lpItemID );
```

Note that this function cannot be used to remove an entry in a playlist.

The function takes the following arguments:

Argument	Description
`lDeviceID`	✓ This is a long integer value to specify a connected NOMAD device. The value should be in the range of 1 to the number of devices returned by `GetDeviceCount()` or `GetDeviceCount2()`.
`lpItemID`	✓ This is a pointer to a `DAPSDK_ID` structure of the ID of the target file in NOMAD device.

The functions may return these values:

Return Constant	Description
`S_OK`	The COM function executed successfully.
`DAPSDK_E_ID_INVALID`	The function failed because the Track or Playlist ID does not exist in the device.
`DAPSDK_E_PLAYER_NOT_CONNECTED`	The function failed because no NOMAD device was detected.
`DAPSDK_E_LOAD_COMPONENTS_FAILED`	The function failed because one or more required files failed to load.
`DAPSDK_E_OUT_OF_MEMORY`	The function failed because there is not enough free memory available in the NOMAD player to complete this operation.
`E_FAIL`	The function failed for unspecified reasons.

Any returned values other than these are returned by the COM system.

AddParentItem()

This function creates a new Parent Item in the NOMAD Jukebox. For the moment, this is only used for adding a new playlist. You must specify the playlist's name in the bstrName field of `DAPSDK_ID` structure. The same structure will return the playlist item ID.

Note that this function is not applicable to NOMAD II players.

```
HRESULT AddParentItem( long lDeviceID, Iunknown * lpRootItemID, long *
    lpParentItemID );
```

The function accepts the following arguments:

Argument	Description
`lDeviceID`	⮟ This is a long integer value to specify a connected NOMAD Jukebox device. The value should be in the range of from 1 to the number of devices returned by `GetDeviceCount2()`.
`lpRootItemID`	⮟ This is a pointer to a `DAPSDK_ID` structure containing a Root Item obtained from previous Root Item enumeration process.
`lpParentItemID`	⮟⮝ This is a pointer to a `DAPSDK_ID` structure where the function writes the identifier to this Parent Item. The bstrName field has to be filled with the playlist's name.

The functions may return these values:

Return Constant	**Description**
`S_OK`	The COM function executed successfully.
`DAPSDK_E_PLAYER_NOT_CONNECTED`	The function failed because no NOMAD Jukebox device was detected.
`DAPSDK_E_LOAD_COMPONENTS_FAILED`	The function failed because one or more required files failed to load.
`DAPSDK_E_OUT_OF_MEMORY`	The function failed because there is not enough free memory available in the NOMAD Jukebox to complete this operation.
`DAPSDK_E_UNSUPPORTED`	The function failed because the parent item type specified is not supported for this function.
`E_FAIL`	The function failed for unspecified reasons.

Any returned values other than these are returned by the COM system.

DeleteParentItem()

This function removes a playlist in the NOMAD Jukebox. Note that the tracks specified in the playlist remain in the NOMAD Jukebox.

Note that this function is not applicable to NOMAD II players.

```
HRESULT DeleteParentItem( long lDeviceID, Iunknown * lpParentItemID );
```

The function takes the following arguments:

Argument	Description
`lDeviceID`	✓ This is a long integer value to specify a connected NOMAD Jukebox device. The value should be in the range of from 1 to the number of devices returned by `GetDeviceCount2()`.
`lpParentItemID`	✓ This is a pointer to a `DAPSDK_ID` structure containing a Parent Item obtained from previous Parent Item enumeration process.

The functions may return these values:

Return Constant	Description
`S_OK`	The COM function executed successfully.
`DAPSDK_E_PLAYER_NOT_CONNECTED`	The function failed because no NOMAD Jukebox device was detected.
`DAPSDK_E_LOAD_COMPONENTS_FAILED`	The function failed because one or more required files failed to load.
`DAPSDK_E_OUT_OF_MEMORY`	The function failed because there is not enough free memory available in the NOMAD Jukebox to complete this operation.
`DAPSDK_E_UNSUPPORTED`	The function failed because the item type specified is not supported for this function. For example, trying to rename a non-playlist type is not allowed.
`E_FAIL`	The function failed for unspecified reasons.

Any returned values other than these are returned by the COM system.

RenameParentItem()

This function renames a parent item in the NOMAD Jukebox. For the moment, this is only applicable to playlists.

Note that this function is not applicable to NOMAD II players.

```
HRESULT RenameParentItem( long lDeviceID, Iunknown *lpParentItemID,
    BSTR bstrNewParentItemName);
```

The function accepts the following arguments:

Argument	**Description**
`lDeviceID`	⋎ This is a long integer value to specify a connected NOMAD Jukebox device. The value should be in the range of from 1 to the number of devices returned by `GetDeviceCount2()`.
`lpParentItemID`	⋎ This is a pointer to a `DAPSDK_ID` structure containing a Parent Item obtained from previous Parent Item enumeration process.
`bstrNewParentItemName`	This is a string of type BSTR containing the new name of the parent item. Note that the COM server will free the memory allocated for `bstrNewParentItemName` at the end of this function.

The functions may return these values:

Return Constant	**Description**
`S_OK`	The COM function executed successfully.
`DAPSDK_E_PLAYER_NOT_CONNECTED`	The function failed because no NOMAD Jukebox device was detected.
`DAPSDK_E_LOAD_COMPONENTS_FAILED`	The function failed because one or more required files failed to load.
`DAPSDK_E_OUT_OF_MEMORY`	The function failed because there is not enough free memory available in the NOMAD Jukebox to complete this operation.
`DAPSDK_E_UNSUPPORTED`	The function failed because the item type specified is not supported for this function. For example, trying to delete a non-playlist type is not allowed.
`E_FAIL`	The function failed for unspecified reasons.

Any returned values other than these are returned by the COM system.

AddItemsToParentItem()

This function adds items to an existing parent item in the NOMAD Jukebox. For the moment, this is only meant to add audio tracks to an existing playlist.

Note that this function is not applicable to NOMAD II players.

```
HRESULT AddItemsToParentItem( long lDeviceID, Iunknown *
    lpParentItemID, long lItemIDCount, Iunknown * lpItemIDList );
```

Note that before you call this function, you must first obtain a valid parent item (or playlist) ID by enumerating the list of parent items in NOMAD Jukebox by calling `FindFirstParentItem()` and `FindNextParentItem`.

The function accepts the following arguments:

Argument	Description
`lDeviceID`	✓ This is a long integer value to specify a connected NOMAD Jukebox device. The value should be in the range of from 1 to the number of devices returned by `GetDeviceCount2()`.
`lpParentItemID`	✓ This `DAPSDK_ID` structure specifies the ID of the parent item (playlist) in NOMAD Jukebox.
`lItemIDCount`	✓ This is a long integer value to specify the number of items pointed to by `lpItemIDList`.
`lpItemIDList`	✓ This is a pointer to an array of `DAPSDK_ID` structures specifying the list of track IDs to be added to the playlist.

The functions may return these values:

Return Constant	Description
`S_OK`	The COM function executed successfully.
`DAPSDK_E_PLAYER_NOT_CONNECTED`	The function failed because no NOMAD Jukebox device was detected.
`DAPSDK_E_LOAD_COMPONENTS_FAILED`	The function failed because one or more required files failed to load.
`DAPSDK_E_OUT_OF_MEMORY`	The function failed because there is not enough free memory available in the NOMAD Jukebox to complete this operation.
`DAPSDK_E_UNSUPPORTED`	The function failed because the item type specified is not supported for this function. For example, non-audio files cannot be added to a playlist.
`E_FAIL`	The function failed for unspecified reasons.

PlayControl()

This is a function new to the SDK, and is used to control the playback operation of a NOMAD Jukebox remotely. This includes starting, pausing, stopping playback, and setting the playback position of an audio track.

Note that this function is not applicable to NOMAD II players.

```
HRESULT PlayControl( long lDeviceID, long lPlayOperationType, Iunknown
    * lpValue );
```

Note that only one track can be played at a time. While the playback is going on, a call to PlayControl() with a new track ID, will stop the current playback and the new one will start.

During the playback, the caller will receive a number of callback messages to notify the caller about progress, completion, or error.

The function accepts the following arguments:

Argument	**Description**
`lDeviceID`	➢ This is a long integer value to specify a connected NOMAD Jukebox device. The value should be in the range of from 1 to the number of devices returned by `GetDeviceCount2()`.
`lPlayOperationType`	➢ This is a long integer value to specify the operation to be performed in the NOMAD Jukebox.
`lpValue`	➢ This value depends on the lPlayOperationType.

The possible values of lPlayOperationType are:

Argument	**Description**
`kPlayTrack`	Commands the NOMAD Jukebox to start playing a particular track from the beginning. lpValue is a pointer to a `DAPSDK_ID` structure of an audio track ID.
`kStopTrack`	Commands the NOMAD Jukebox to stop the current playback. lpValue is not used and should be set to NULL.
`kPauseTrack`	Commands the NOMAD Jukebox to pause the current playback. lpValue is not used and should be set to NULL.
`kSetPlaybackPosition`	Commands the NOMAD Jukebox to move the current playback to a particular point in the file. This will immediately cause the playback to resume from that point. lpValue is a pointer to a long integer containing the position in percentage of the file.

These are the possible callback messages:

Message	**Description**
`WM_DAPSDK_PLAYBACK_PROGRESS`	Notifies the caller of playback progress. wParam: percent complete, lParam: unused.
`WM_DAPSDK_PLAYBACK_COMPLETE`	Notifies the caller that the playback is completed successfully. wParam, lParam unused.
`WM_DAPSDK_PLAYBACK_ERROR`	Notifies the caller that the playback has encountered an error during playback and has stopped. wParam: error code, lParam: unused.

The functions may return these values:

Return Constant	Description
`S_OK`	The COM function executed successfully.
`DAPSDK_E_PLAYER_NOT_CONNECTED`	The function failed because no NOMAD Jukebox device was detected.
`DAPSDK_E_LOAD_COMPONENTS_FAILED`	The function failed because one or more required files failed to load.
`DAPSDK_E_OUT_OF_MEMORY`	The function failed because there is not enough free memory available in the NOMAD Jukebox to complete this operation.
`DAPSDK_E_UNSUPPORTED`	The function failed because the item type specified is not supported for this function. Most likely, the item ID does not belong to an audio file.
`E_FAIL`	The function failed for unspecified reasons.

QueueControl()

This is a function new to the SDK and is used to supplement the playback operation by adding and removing a single track to a playback queue in the NOMAD Jukebox. When the current playing track is completed, the queued track will start playback immediately.

Note that this function is not applicable to NOMAD II players.

```
HRESULT QueueControl( long lDeviceID, long lQueueOperationType,
    Iunknown * lpValue );
```

You can achieve some form of continuous playback system of multiple tracks by starting one playback, queuing another one, and whenever the ..._PLAYBACK_COMPLETE message is received, queue another track for playback.

This is a synchronous operation and no callback is required.

The function accepts the following arguments:

Argument	Description
`lDeviceID`	✓ This is a long integer value to specify a connected NOMAD Jukebox device. The value should be in the range of from 1 to the number of devices returned by `GetDeviceCount2()`.
`lQueueOperationType`	✓ This is a long integer value to specify the operation to be performed in the NOMAD Jukebox.
`lpValue`	✓ This value depends on `lQueueOperationType`.

The possible values of lQueueOperationType are:

Argument	**Description**
`kQueueTrack`	Adds an audio track to playback queue in the NOMAD Jukebox. lpValue is a pointer to a `DAPSDK_ID` structure of an audio track ID.
`kClearQueue`	Removes the previously queued item in the playback queue in the NOMAD Jukebox. lpValue is not used and should be set to NULL.

The functions may return these values:

Return Constant	**Description**
`S_OK`	The COM function executed successfully.
`DAPSDK_E_PLAYER_NOT_CONNECTED`	The function failed because no NOMAD Jukebox device was detected.
`DAPSDK_E_LOAD_COMPONENTS_FAILED`	The function failed because one or more required files failed to load.
`DAPSDK_E_OUT_OF_MEMORY`	The function failed because there is not enough free memory available in the NOMAD Jukebox to complete this operation.
`DAPSDK_E_UNSUPPORTED`	The function failed because the item type specified is not supported for this function. Most likely, the item ID does not belong to an audio file.
`E_FAIL`	The function failed for unspecified reasons.

NOMAD SDK Windows Messages

The NOMAD library uses pre-defined window messages to notify the application of progress and device connection states. Applications who wish to receive such notifications have to register with the SDK using the function `SetCallbackWindow2()` (or `SetCallbackWindow()` for NOMAD II), passing in the window handle as argument.

A window receives this message through its WindowProc function.

```
LRESULT CALLBACK WindowProc(

    HWND hwnd,        // handle to window

    UINT uMsg,        // message IDHANGE

    WPARAM wParam,     // depends on message ID

    LPARAM  lParam

);
```

The following are messages returned by the SDK:

WM_DAPSDK_GETITEM_PROGRESS

This message notifies the application of the file transfer progress. This message is returned in response to a `GetItem()` call. The return value is a percentage number (0-100%). The arguments are:

Argument	Description
`wParam`	This is an integer value to specify the percentage of completion of the current file transfer.
`lParam`	Currently unused.

WM_DAPSDK_GETITEM_COMPLETE

This message notifies the application of the file transfer progress. This message is returned in response to a `GetItem()` call. The arguments are:

Argument	Description
`wParam`	An error status of the completed process.
`lParam`	Currently unused.

WM_DAPSDK_PLAYBACK_PROGRESS

This message notifies the application of the current playback progress in NOMAD Jukebox. This message is returned in response to a `PlayControl(..,kPlayTrack,..)` call. The return value is a percentage number (0-100%). The arguments are:

Argument	Description
`wParam`	The current playtime in seconds.
`lParam`	Currently unused.

WM_DAPSDK_ PLAYBACK_COMPLETE

This message notifies the application that the current track has completed playback in NOMAD Jukebox. This message is returned in response to a `PlayControl(..,kPlayTrack,..)` call. The arguments are:

Argument	Description
`wParam`	Currently unused.
`lParam`	Currently unused.

WM_DAPSDK_ PLAYBACK_ERROR

This message notifies the application that an error has occurred during the current playback and the process has stopped in NOMAD Jukebox. This message is returned in response to a `PlayControl(..,kPlayTrack,..)` call. The arguments are:

Argument	Description
`wParam`	An error status of the playback process.
`lParam`	Currently unused.

WM_DAPSDK_ADDITEM_PROGRESS

This message notifies the application of the file transfer progress. The return value is a percentage number (0-100%). The arguments are:

Argument	Description
`wParam`	This is an integer value to specify the percentage of completion of the current file transfer.
`lParam`	Currently unused.

WM_DAPSDK_ADDITEM_COMPLETE

This message notifies the application of the end of a file transfer progress. The arguments are:

Argument	Description
`wParam`	An error status of the completed process.
`lParam`	This is a pointer to an unsigned long integer containing the ID of the newly created item. This ID is the lID member of `DAPSDK_ID` structure.

WM_DAPSDK_JUKEBOX_REMOVAL

This message notifies the application that a NOMAD Jukebox has just been removed from the computer. Currently unsupported for NOMAD II devices. The arguments are:

Argument	Description
`wParam`	Currently unused.
`lParam`	Currently unused.

WM_DAPSDK_JUKEBOX_ARRIVAL

This message notifies the application that a new NOMAD Jukebox was just connected to the computer. Currently unsupported for NOMAD II devices. The arguments are:

Argument	Description
`wParam`	Currently unused.
`lParam`	Currently unused.

Structure Summary

The following section describes the structures for exchanging data with Nomad devices in most of the function calls.

DAPSDK_ID

This structure describes the item identifier in this SDK. This structure is the same for all items, root items and parent items.

```
typedef struct _DAPSDK_ID

{

    long lID;

    long lType;

    BSTR bstrName;

} DAPSDK_ID, *PDAPSDK_ID;
```

The descriptions of the arguments follow:

Argument	Description
`lID`	Unique file handle
`lType`	For NOMAD Jukebox, file type, kAudioTrackType, kPlaylistType, or kDataFileType. For NOMAD Jukebox 2/3/Zen, also kDataFolderType. For NOMAD II, memory type, either kInternalMemType or kExternalMemType.
`bstrName`	Item name string.

DAPSDK_VERSION

This structure describes the version information used in this SDK. This structure is used in `GetSDKVersion`() and `GetDeviceProperties(kFirmwareVersion)` calls.

```
typedef struct _DAPSDK_VERSION

{

    WORD major;

    WORD minor;

    WORD build;

    WORD specialBuild;

} DAPSDK_VERSION, *PDAPSDK_VERSION;
```

The descriptions of the arguments follow:

Argument	Description
`major`	Major version number – indicating major changes to API.
`minor`	Minor version number – indicating minor update to SDK.
`build`	Build number – mostly for bug fix update.
`specialBuild`	Special build number – currently unused.

DAPSDK_RADIOPRESET

This structure is used in `SetDeviceProperties(kFMRadioPresetValue)` call to set the frequency of an FM radio preset in NOMAD II.

```
typedef struct _DAPSDK_RADIOPRESET

{

    DWORD dwPresetIndex;

    DWORD dwPresetValue;

} DAPSDK_RADIOPRESET, *PDAPSDK_RADIOPRESET;
```

The descriptions of the arguments follow:

Argument	Description
`dwPresetIndex`	Index of the radio preset (0 to 31).
`dwPresetValue`	Frequency value in KHz (e.g., 103.3 MHz is 103300).

DAPSDK_DEVICE_INFO

This structure is used in GetDeviceInfo`()` call in ICTJukeBox interface.

```
typedef struct _DAPSDK_DEVICE_INFO

{

    BYTE cDeviceId[16];

     long dwFirmwareVersion;

     long dwHardwareVersion;

    char cDeviceName[32];

    BYTE byPowerSource;

} DAPSDK_DEVICE_INFO, *PDAPSDK_DEVICE_INFO;
```

The descriptions of the arguments follow:

Argument	Description
`cDeviceID`	128-bit device serial number of the NOMAD Jukebox.
`dwFirmwareVersion`	Firmware version information in the form: Major(8):Minor(8):Build(16).
`dwHardwareVersion`	Hardware version information in the form: Major(8):Minor(8):Build(16).
`cDeviceName`	Descriptive name of Jukebox in ASCII.
`byPowerSource`	Battery level in percentage.

DAPSDK_STORAGE_INFO

This structure describes the storage space information in the NOMAD device. It is returned by the call `GetDeviceProperties( kStorageInfoStruct)`. Note that the numbers are in high and low parts of 64-bit integers and the developer is expected to combine them correctly into a single 64-bit integer.

```
typedef struct _DAPSDK_STORAGE_INFO

{

    ULONG   totalH;

    ULONG   totalL;

    ULONG   freeH;

    ULONG   freeL;

} DAPSDK_STORAGE_INFO, *PDAPSDK_STORAGE_INFO;
```

The descriptions of the arguments follow:

Argument	Description
`totalH`	High part of 64-bit integer describing the total available storage space.
`totalL`	Low part of 64-bit integer describing the total available storage space.
`freeH`	High part of 64-bit integer describing the available disk space storage use.
`freelL`	Low part of 64-bit integer describing the available storage space for use.

FormatInfo

This structure describes the format details of a CODEC supported in the NOMAD player. This structure is used in `FindFirstFormatSupport()` and `FindNextFormatSupport()`.

```
typedef struct _LIB_FORMAT_SUPPORT

{

    long CodecID;

    BYTE CodecName[MAX_CODECNAME_Len];

    long lMaxSamplingRate;

    long lMaxNumChannels;

} FORMATINFO, * PFORMATINFO;
```

The descriptions of the arguments follow:

Argument	**Description**
`CodecID`	A long integer value to identify this CODEC.
`CodecName`	Descriptive string for this CODEC.
`lMaxSamplingRate`	Maximum sample rate supported for this CODEC in Hz.
`lMaxNumChannels`	Maximum number of channels supported for this CODEC.

DateTime

This structure describes the date and time information in the NOMAD player. This structure is used in `GetDateTime()` and `SetDateTime()`.

```
typedef struct _LIB_DATE_TIME

{

    short sYear;

    short sMonth;

    short sDay;

    short sDayofWeek;

    short sHour;

    short sMinutes;

    short sSeconds;

} DATETIME, * PDATETIME;
```

ItemInfo

This structure describes the attributes of a track of the NOMAD device. This is a variable-length structure with this format:

Field	Offset	Size	Description
`Attribute Count`	0	2	The number of attributes in this structure.
`Attribute 1 type`	2	2	ASCII=0, Binary=1, UNICODE=2.
`Attribute 1 name length`	4	2	Length of name string in bytes.
`Attribute 1 data length`	6	4	Length of attribute data in bytes.
`Attribute name`	10	N	Attribute name string.
`Attribute data`	10+N	M	Attribute data.

For NOMAD Jukebox, an item attribute packet can have several attributes describing the details of a `kAudioTrackType` item, similar to those defined in the ID3 tag of an MP3 file. The following is a list of attributes currently defined:

Attribute Name	**Attribute Type**	**Possible Values**
`TITLE`	`ASCII/UNICODE`	Text string.
`CODEC`	`ASCII`	"MP3", "WMA", or "WAV".
`ALBUM`	`ASCII/UNICODE`	Text string.
`ARTIST`	`ASCII/UNICODE`	Text string.
`FILE SIZE`	`BINARY`	Unsigned long value in bytes.
`GENRE`	`ASCII/UNICODE`	Text string.
`LENGTH`	`BINARY`	Unsigned long value in seconds.
`TRACK ID`	`BINARY`	Unsigned long value.
`TRACK NUM`	`BINARY`	Unsigned short value.
`YEAR`	`BINARY`	Unsigned short year value.

For NOMAD Jukebox 2/3/Zen, the following attributes are additional:

Attribute Name	**Attribute Type**	**Possible Values**
`FILE ATTRIB`	`BINARY`	Unsigned long corresponding to Windows API File Attributes.
`PARENT FOLDER`	`BINARY`	Unsigned long value containing the `lID` part of `DAPSDK_ID` of a data folder item.
`FOLDER NAME`	`ASCII/UNICODE`	Text string for data folder name.
`FILE NAME`	`ASCII/UNICODE`	Text string for data file name.
`MODIFIED FILETIME`	`BINARY`	Unsigned long value of file time.

For NOMAD II, an item attribute packet is fixed to only the following:

Attribute Name	Attribute Type	Possible Values
`FILE NAME`	`ASCII`	Text string.
`FILE SIZE`	`BINARY`	Unsigned long value in bytes.
`DOS_DATETIME`	`BINARY`	32-bit value.
`DOS_FILEATTRIB`	`BINARY`	Unsigned char value.

ICTJukeBox/ICTNomad2 Function Summary

The following table lists `ICTJukeBox`'s and `ICTNomad2`'s member functions along with the prototype for each function. The functions are listed in dependency order from lowest level to highest level. For more information on function dependency, see the section titled Using NOMAD API Functions earlier in this document.

Member Function	Function Prototype
`AddItem`	`HRESULT AddItem( long` **`lDeviceID`**`, long lItemType, BSTR` **`bstrSrcFileName`**`, long` **`lItemInfoSize`**`, Iunknown *` **`lpItemInfo`**`);`
`AddItemsToParentItem`	`HRESULT AddItemsToParentItem( long` **`lDeviceID`**`, long` **`lItemCount`**`, void *` **`lpItemIDList`**`, Iunknown *` **`lpParentItemID`** `);`
`AddParentItem`	`HRESULT AddParentItem( long` **`lDeviceID`**`, Iunknown *` **`lpParentItemID`** `);`
`DeleteParentItem`	`HRESULT DeleteParentItem( long` **`lDeviceID`**`, Iunknown *` **`lpParentItemID`**`);`
`DeleteItem`	`HRESULT DeleteItem( long` **`lDeviceID`**`, Iunknown` **`*`** **`lpItemID`** `);`
`FindFirstParentItem`	`HRESULT FindFirstParentItem( long` **`lDeviceID`**`, Iunknown *` **`lRootItemID`**`, Iunknown *` **`lpParentItemID`** `);`

Member Function	Function Prototype
FindFirstItem	HRESULT FindFirstItem(long **lDeviceID**, Iunknown * **lpParentItemID**, Iunknown * **lpItemID**);
FindNextParentItem	HRESULT FindNextParentItem(long **lDeviceID**, Iunknown * **lRootItemID**, Iunknown * **lpParentItemID**);
FindNextRootItem	HRESULT FindNextRootItem(long **lDeviceID**, Iunknown * **lpRootItemID**, BSTR * **lpRootItemName**);
FindNextItem	HRESULT FindNextItem(long **lDeviceID**, Iunknown * **lpParentItemID**, Iunknown * **lpItemID**);
GetDeviceCount	HRESULT GetDeviceCount(long* **lpCount**);
GetDeviceCount2	HRESULT GetDeviceCount2(long* **lpCount**);
GetDeviceProperties	HRESULT GetDeviceProperties (long **lDeviceID**, long **lPropertyType**, Iunknown * **lpValue**);
GetSDKVersion	HRESULT GetSDKVersion (long* **lpSDKVersion**);

Member Function	Function Prototype
GetSDKVersion2	HRESULT GetSDKVersion2 (long* **lpSDKVersion**);
GetItemAttribute	HRESULT GetItemAttribute(long **lDeviceID**, Iunknown * **lpItemID**, long **lInItemInfoSize**, long * **lOutItemInfoSize**, Iunknown * **lpItemInfo**);
GetItem	HRESULT GetItem(long **lDeviceID**, BSTR **bstrDestinationFileName**, IUnknown * **lpItemID**);
Initialize	HRESULT Initialize();
Initialize2	HRESULT Initialize2();
PlayControl	HRESULT PlayControl (long **lDeviceID**, long **lPlayOperationType**, Iunknown * **lpValue**);
QueueControl	HRESULT QueueControl (long **lDeviceID**, long **lQueueOperationType**, Iunknown * **lpValue**);
RenameParentItem	HRESULT RenameParentItem(long **lDeviceID**, Iunknown * **lpParentItemID**, BSTR **bstrNewParentItemName**);
SetCallbackWindow	HRESULT SetCallbackWindow(long **lDeviceID**, long **hWnd**);

Member Function	**Function Prototype**
SetCallbackWindow2	HRESULT SetCallbackWindow2(long **lDeviceID**, long **hWnd**);
SetDeviceProperties	HRESULT SetDeviceProperties (long **lDeviceID**, long **lPropertyType**, Iunknown * **lpValue**);
SetItemAttribute	HRESULT SetItemAttribute(long lDeviceID, Iunknown * lpItemID, BSTR bstrAttributeName, long lAttributeType, long lAttributeDataSize, IUnknown *lpAttributeData);
Shutdown	HRESULT Shutdown();
Shutdown2	HRESULT Shutdown2();

Error Code Reference

ICTJukeBox, ICTJukeBox2, and ICTNomad2 member functions return an HRESULT value. If execution was successful, the value is S_OK. If unsuccessful, it is an error code defined as a constant in the file nmsdk.h.

The following table shows, in alphabetic order, the possible HRESULT error constants generated by ICTJukeBox, ICTJukebox2, and ICTNomad2. An explanation of the error accompanies each constant. The chart also shows text your application should display to the user if it is appropriate for the error.

Note that the functions may return error codes that are not defined here. These are generated by the COM system; you can find explanations for them in COM documentation.

Error Code	Explanation
DAPSDK_E_AUDIOFILE_FORMAT	The function failed because the file to transfer into NOMAD Jukebox contains an unsupported audio format.
DAPSDK_E_AUDIOFILE_INVALID	The function failed because the file to transfer into NOMAD Jukebox appears to be corrupted.
DAPSDK_E_CODEC_NOT_SUPPORTED	The specified track cannot be played because the player does not have the CODEC.
DAPSDK_E_DATA_CORRUPTED	The specified track cannot be played because the file is corrupted.
DAPSDK_E_DATA_FILE_ALREADY_EXIST	The function failed because a file of the same name already exists in the player.
DAPSDK_E_DATA_FILE_NOT_FOUND	The function failed because the specified path data file item is not valid.
DAPSDK_E_DATA_FILE_TOO_BIG	The specified data file size exceeds the maximum file size allowed.
DAPSDK_E_DECODING_ERROR	The playback is aborted because the player encountered an error when decoding the file.
DAPSDK_E_DISKFULL_FOR_DOWNLOAD	The function failed because the disk space NOMAD Jukebox is fully consumed.

Error Code	Explanation
`DAPSDK_E_END_OF_LIST`	Playback of a playlist has ended. This is a status message returned from the SDK.
`DAPSDK_E_END_OF_TRACK`	Playback of a track has ended. This is a status message returned from the SDK.
`DAPSDK_E_FILE_READ_WRITE_FAILED`	The function failed because the library DLL cannot continue copying the file into the PC.
`DAPSDK_E_FILETYPE_ILLEGAL`	The function failed because the item type is not one of those defined in the SDK.
`DAPSDK_E_FORMAT_NOT_FOUND`	The function failed because there are no more audio formats to find. End of enumeration.
`DAPSDK_E_ID_INVALID`	The function failed because the Track or Playlist ID does not exist in the device. This error may also be returned if the device ID specified is not valid or the device is no longer connected.
`DAPSDK_E_ITEM_NOT_FOUND`	The function failed because there are no more items to find. End of enumeration.
`DAPSDK_E_ITEM_UPLOAD_DENIED`	The function failed because the item (file) cannot be uploaded due to copy protection.
`DAPSDK_E_LIB_BUSY`	The function failed because the NOMAD library DLL is currently busy with previous operation.
`DAPSDK_E_LIB_CORRUPTED`	The function failed because the database in the NOMAD is corrupted. Users should do a disk cleanup by re-starting the player into maintenance mode.
`DAPSDK_E_LOAD_COMPONENTS_FAILED`	The function failed because one or more required files failed to load.

Error Code	Explanation
`DAPSDK_E_NO_STORAGE`	This function failed because you attempted to access files in a non-existent storage media. For example, NOMAD II without internal memory or NOMAD II MG without a SmartMedia card inserted.
`DAPSDK_E_NORIGHTS`	This function failed because the file cannot be transferred due to DRM protection.
`DAPSDK_E_NORIGHTS`	The function failed because the system does not have the right to handle this DRM protected file. Either the system does not have the license, or the rights specified in the license do not include performing the requested operation (such as transfer to player).
`DAPSDK_E_OUT_OF_MEMORY`	The function failed because there is not enough free memory available in the NOMAD player to complete this operation.
`DAPSDK_E_PARENTNODE_NOT_EXIST`	The function failed because the parent item specified in the function argument does not exist in the player.
`DAPSDK_E_PATH_EXCESS_LEN`	The function failed because the file path specified is too long to be stored in the player.
`DAPSDK_E_PLAYBACK_INPROGRESS`	The function failed because the player is currently in playback mode. File operations cannot proceed.
`DAPSDK_E_PLAYER_NOT_CONNECTED`	The function failed because no NOMAD device was detected.
`DAPSDK_E_PORT_UNAVAILABLE`	The function failed because the USB/1394 port is in use.
`DAPSDK_E_POSITION_OUTOF_RANGE`	The function failed because the playback position specified is out of the range of the track duration.
`DAPSDK_E_SAMPLING_RATE_NOT_SUPPORTED`	The specified track cannot be played because the player does not support the sample rate.

Error Code	Explanation
DAPSDK_E_SMARTMEDIA_WRITE_PROTECTED	This function failed because you attempted to add or delete files into a write-protected SmartMedia card.
DAPSDK_E_STATUS_TIMEOUT	The function failed because the player has failed to respond after a period of time. Either the player was disconnected or is not functioning.
DAPSDK_E_TOO_MANY_DATA_FILES	The function failed because the number of data file items in the player has reached the limit.
DAPSDK_E_TRACK_ALREADY_EXIST	The function failed because a track with the same Album, Artist, and Title already exists in the NOMAD Jukebox or a file with the same name already exists in the NOMAD II player.
DAPSDK_E_TRACK_READ_WRITE_ERROR	The function failed because an error occurred during the file transfer process.
DAPSDK_E_TRANSFER_INPROGRESS	The function failed because the player is currently transferring a file. Playback operations cannot proceed.
DAPSDK_E_UNSUPPORTED	The function failed because the item type or property type specified is not supported for this function.
DAPSDK_E_WMDM_INIT_FAILED	The function failed because Windows Media Device Manager (WMDM) failed to initialize.
E_FAIL	The function failed for unspecified reasons.
E_INVALIDARG	The function failed to execute properly. One or more arguments are invalid or a NULL pointer.
S_OK	The COM function executed successfully.